НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри
_____ Сергій СТІРЕНКО

“___”_____2020 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп’ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Система обробки транзакцій в мережі криптовалюти Ethereum»

Виконав:
студент IV курсу, групи ІО-62
Дарагань Денис Андрійович _____

Керівник:
професор, д.т.н.,
Новотарський Михайло Анатолійович _____

Консультант з нормоконтролю:
професор, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Доцент кафедри СКС ФПМ, к.т.н.
Орлова Марія Миколаївна _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврьский)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___» _____2020 р.

ЗАВДАННЯ
на дипломний проект студента

Дараганя Дениса Андрійовича

1. Тема проекту «Система обробки транзакцій в мережі криптовалюти Ethereum» керівник проекту Новотарський Михайло Анатолійович, д.т.н., професор,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «07» травня 2020 року №1081-с
2. Термін здачі студентом закінченого проекту

_____
3. Вихідні дані до проекту *технічна документація*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Проектування системи обробки транзакцій в мережі криптовалюти Ethereum та розробка пов'язаного користувацького інтерфейсу. Опис предметної області, дослідження засобів розробки програмного забезпечення, розробка алгоритмів для вирішення задач прийому та обробки інформації, реалізація роботи з БД та інтерфейсами публічних сервісів, програмна реалізація та тестування.
5. Перелік графічного матеріалу
Принципова схема, функціональна схема та структурна схема_____

6. Консультанти проекту, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
нормоконтроль	д.т.н., проф. Сімоненко В.П.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту	Строк виконання етапів проекту	Примітки
1.	*Затвердження теми роботи*	*1.09.2019*	
2.	*Вивчення та аналіз завдання*	*2.09.2019-15.03.2020*	
3.	*Розробка архітектури та загальної структури програми*	*15.03.2020-25.03.2020*	
4.	*Розробка структур окремих інтерфейсів програми*	*25.03.2020-5.04.2020*	
5.	*Програмна реалізація*	*5.04.2020-15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04.2020-20.05.2020*	
7.	*Захист програмного продукту*	*21.05.2020 – 25.05.2020*	
8.	*Передзахист*	*26.05.2020*	
9.	*Захист*		

Студент Денис ДАРАГАНЬ

Керівник Михайло НОВОТАРСЬКИЙ

Анотація

Дана робота – «Система обробки транзакцій в мережі криптовалюти Ethereum» – присвячена популяризації криптовалютних систем. Проблематикою для вирішення обрано необхідність спрощення взаємодії платника із віртуальними активами за допомогою створення нативного інтерфейсу. Причиною вибору даного напрямку є зростання популярності криптовалют, що обумовлено їхніми перевагами.

Робота пропонує користувацький інтерфейс, представлений віртуальним співрозмовником месенджеру Telegram, а також алгоритм, за допомогою якого відбувається взаємодія між програмою та мережею криптовалюти. Перевагою розробки є її невибагливість до ресурсів комп'ютерної системи, на якій вона розгортається.

Аннотация

Данная работа – «Система обработки транзакций в сети криптовалюты Ethereum» – посвящена популяризации криптовалютных систем. Проблематикой для решения выбрано необходимость упрощения взаимодействия плательщика с виртуальными активами при помощи создания нативного интерфейса. Причиной выбора данного направления есть рост популярности криптовалют, что обусловлено их преимуществами.

Работа предлагает пользовательский интерфейс, представленный виртуальным собеседником мессенджера Telegram, а также алгоритм, при помощи которого происходит взаимодействие между программой и сетью криптовалюты. Преимуществом разработки есть её неприхотливость к ресурсам компьютерной системы, на которой она разворачивается.

Annotation

This work – “Transaction Processing System in the Ethereum Cryptocurrency Network” – is devoted to the population of cryptocurrency systems. The problematic for the solution was the need to simplify the interaction between the payer and virtual assets by creating a native interface. The reason for choosing this direction is the growing popularity of cryptocurrencies, because of their advantages.

The work offers a user interface presented by the Telegram messenger virtual chatter and an algorithm by which the interaction between the program and the cryptocurrency network occurs. The advantage of development is its unpretentiousness to the resources of the computer system on which it is deployed.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Ф	Позначення	Найменування	лис Кіль	Примітка
1.	А4		Завдання на дипломний проект	2	
2.	А4	ДП.7800.02.000 ТЗ	Технічне завдання	5	
3.	А4	ДП.7800.03.000 ПЗ	Пояснювальна записка	67	
4.	А3	ДП.7800.04.000 А1	Принципова схема алгоритму	1	
5.	А4	ДП.7800.05.000 А2	Функціональна схема	1	
6.	А4	ДП.7800.06.000 А3	Структурна схема	1	
7.	А4	ДП.7800.07.000 Б1	Лістинг програми	8	

					ДП.7800.01.000 ВП		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Дарагань Д.А.			Система обробки транзакцій в мережі криптовалюти Ethereum Відомість дипломного проекту	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Новотарський М.А.					
Реценз.						НТУУ «КПІ», ФІОТ, ІО-62	
Н. Контр.		Сімоненко В.П.					
Затв.							

Технічне завдання
до дипломного проекту

на тему: «Система обробки транзакцій в мережі криптовалюти Ethereum»

Київ – 2020

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування розробки: «Система обробки транзакцій в мережі криптовалюти Ethereum».

Область застосування розробки: програма знаходиться у вільному доступі та може бути використана будь-ким з метою отримування та перегляду інформації стосовно певних процесів, що протікають у мережі цільової криптовалюти.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка системи сповіщень про нові події мережі цільової криптовалюти, яка взаємодіє з користувачем за допомогою власного інтерфейсу.

4. ДЖЕРЕЛА РОЗРОБКИ

У якості джерел розробки було використано:

- науково-технічну літературу;
- публікації в спеціалізованих періодичних виданнях та мережі Інтернет;
- документацію API сервісів Telegram Bot, Etherscan, INFURA;
- документацію фреймворків aiogram, aio-pika, aiohttp.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту

Розроблювана система має задовольняти наступні потреби:

- Запам’ятовування адрес мережі цільової криптовалюти;
- Пошук та відображення записів історії транзакцій за збереженими адресами;
- Сповіщення про нові транзакції збережених адрес;
- Налаштування відображення інформації;
- Конкуренто спроможна швидкість оновлення даних;
- Невибагливість до ресурсів комп’ютерних систем.

5.2. Вимоги до апаратної частини

Система для роботи потребує серверне та клієнтське апаратне забезпечення.

Мінімальні вимоги до серверної сторони:

- UNIX-подібна операційна система (бажано дистрибутив Ubuntu 16.04);
- 1 гігабайт оперативної пам’яті;
- 1 гігабайт пам’яті жорсткого диску;
- 1 ядро процесору з тактовою частотою не менше 2 ГГц;
- З’єднання з мережею Інтернет зі швидкістю не менше 50 Мбіт/с.

Мінімальні вимоги до клієнтської сторони:

- Пристрій зі встановленим додатком Telegram Messenger або доступом до веб-версії Telegram.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення та аналіз завдання	15.03.2020
Розробка архітектури програми	25.03.2020
Розробка інтерфейсів програми	05.04.2020
Програмна реалізація	15.04.2020
Тестування окремих модулів системи	25.04.2020
Доопрацювання, налагодження і виправлення помилок	30.04.2020
Оформлення документації дипломної роботи	20.05.2020

Пояснювальна записка
до дипломного проекту

на тему: «Система обробки транзакцій в мережі криптовалюти Ethereum»

Київ – 2020

ЗМІСТ

ВСТУП

Із розвитком цифрових технологій поступово збільшувався суспільний інтерес до нових можливостей людини. Стрімко почала зростати світова сфера IT. У компаніях, більшість з яких щойно вийшли на ринок, виникла потреба у великій кількості робітників-програмістів.

На сьогодні ринок спеціалістів в області IT перенасичений. Пропозиція кадрів у сфері програмування перевищує попит. Це явище тісно пов’язано зі збільшенням світової інфраструктури цифровими засобами для обміну інформацією. Тобто, на даний момент наявний великий коефіцієнт насичення цифровими даними найбільш розповсюдженої мережі Інтернет.

На фоні вільного доступу до більшості джерел інформації виникає потреба виключення можливості перегляду чи редагування сторонніми особами певного ряду приватних даних, що також знаходяться в мережі Інтернет. Це призвело до виокремлення нового напрямку в сфері розвитку технологій – захисту інформації.

Одним з найбільш чутливих до необхідності забезпечення приватності інформації сегментів мережі є Інтернет-банкінг. Компрометація ключів алгоритму шифрування, на якому базується робота певного банку означає повний доступ до всіх активів, що йому належать. Це стало причиною стрімкого розвитку та популяризації децентралізованих платіжних систем, що відзначаються високим рівнем захищеності. На відміну від стандартних систем, вони поєднують використання криптографічних методів захисту інформації та велику кількість копій усіх даних, що їм належать. Такий спосіб роботи для компрометації даних потребує не існуючої на даний момент комп’ютерної потужності від зловмисника.

Децентралізовані платіжні системи працюють із цифровими активами, які прийнято називати криптовалютами. Найбільш відомими та розповсюдженими криптовалютами вважають Bitcoin та Ethereum.

Актуальність проектної роботи полягає у взаємодії з набираючою популярність мережею криптовалюти Ethereum.

Одним з розповсюджених недоліків криптовалютних систем є відсутність зручного стандартного інтерфейсу для взаємодії зі своїми активами в даній мережі. Отже, існує проблема оптимізації процесів взаємодії користувача та популярної криптовалюти.

Метою роботи є популяризація децентралізованих платіжних систем серед користувачів мережі завдяки створенню доступного інтуїтивно зрозумілого інтерфейсу.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні відомості про криптовалюту

Криптовалюта – це особливий вид цифрових грошей, одиницею якого є coin (монета). Префікс крипто- вказує на спосіб захисту інформації – використання криптографічних методів.

Вперше термін криптовалюта було застосовано після появи першої блокчейн мережі з можливістю платіжних операцій під назвою Bitcoin у 2009 році. Ідея швидко здобула розповсюдження, що спричинило появу альткойнів – альтернативних по відношенню до Bitcoin мереж криптовалют. На даний момент налічується більше 1000 подібних блокчейн систем із власними особливостями, перевагами та недоліками.

Криптовалюта характеризується як децентралізований додаток (DApp), тобто вона відповідає наступним критеріям:

- Вихідний код доступний для кожного;
- Базується на криптографічній технології (блокчейн);
- Має цифрові активи (токени);
- Здатна генерувати токени та має алгоритм консенсусу (правила узгодження розподілених даних).

Певна криптовалюта може бути віднесена до одного з трьох типів у залежності від моделі блокчейну, що вона використовує:

1. Має власну блокчейн-платформу;
2. Використовує блокчейн криптовалюти першого типу. Таку криптовалюту називають протоколом;
3. Використовує протокол криптовалюти другого типу [1].

Криптовалюта принципово відрізняється від звичних активів за рядом критеріїв, серед яких можна виділити наступні:

– Децентралізована система зберігання – весь грошовий об’єм розподілений між користувачами мережі;

– Відсутність фізичного вираження – кількісне означення криптовалюти є просто числом у відповідній позиції інформаційного пакету;

– Публічна база транзакцій – усі перекази між адресами мережі знаходяться у відкритому доступі та можуть бути переглянуті на відповідних порталах [2].

Завдяки своїм особливостям криптовалюта перед звичайною валютою має ряд таких переваг як:

– Стійкість до підробки даних – забезпечується шифруванням та децентралізацією, тобто підробити будь-які дані означає зламати алгоритм шифрування та провести необхідну заміну одночасно на всіх комп’ютерах мережі, що потребує не існуючої на даний момент потужності;

– Незалежність – відсутність внутрішнього або зовнішнього органу керування обсягу випуску валюти, а також контролю пересування активів;

– Доступність – скористатися своїми активами можна у будь-який час, у будь-якій країні (якщо в ній не заборонена криптовалюта законодавством) без ризику арешту (блокування) рахунку чи вилучення з нього коштів [3].

На даний момент роль криптовалюти на законодавчому рівні остаточно не визначена в жодній державі, а її розвиток цілком залежить від популяризації та рівня довіри серед платників. Наприклад, у 2017 році курс найпопулярнішої криптовалюти Bitcoin перевищив 20000 доларів США за одну монету [4], що вказує на пік зацікавленості з боку користувачів мережі Інтернет.

1.2. Переваги криптовалют

1.2.1. Анонімність

Однією з найбільш привабливих особливостей криптовалюти є анонімність власника рахунку. Тобто, реєстрація нового користувача полягає у “бронюванні” унікального номера-акаунта в мережі без видачі персональних даних. Це означає, що інші користувачі мережі будуть бачити лише символьну адресу з якої здійснюються транзакції, але не можуть нічого дізнатися про власника цієї адреси.

Відсутність інформації щодо власників рахунків стала причиною розповсюдження криптовалют в мережі Tor. Такі активи виявилися дуже зручними для оплати товарів та послуг в анонімних Інтернет-магазинах. Не зважаючи на те, що криптовалюта була задіяна в незаконних транзакціях, саме ця популярність дала чималий поштовх її розвитку.

Не зважаючи на те, що найбільший обсяг криптовалютних транзакцій залишається в анонімній мережі, з’являються легальні магазини, які приймають віртуальні гроші в якості оплати товарів чи послуг. Наприклад, на даний момент в Україні можна придбати електротехніку, одяг, розрахуватися за каву або доставку квітів монетами Bitcoin [5].

1.2.2. Смарт-контракти

На даний момент найбільш перспективним напрямком розвитку криптовалютних систем є другий тип децентралізованих додатків, так звані розумні контракти – програмний код, який використовує математичні алгоритми для автоматичного виконання певних дій, так чи інакше пов’язаних із оплатою в мережі. Даний протокол широко застосовується в другій за популярністю мережі криптовалюти Ethereum. На основі правил, що закладені в розумних контрактах працюють децентралізовані додатки третього типу, які являють собою користувацькі сервіси [6].

Ідея та принципи роботи розумних контрактів були сформульовані в 1996 році, однак реалізація стала можливою тільки після розвитку децентралізованих блокчейн платформ.

Зміст смарт-контракту полягає в тому, що деякий код надсилається в мережу для реєстрації. Після підтвердження він починає свою роботу або стає доступним для використання. Тобто, коли, наприклад, надходить переказ грошей на адресу, до якої прив'язана дана програма, відбувається перевірка певних умов, описаних у її тілі. Якщо перевірка успішна та всі умови задоволені, то далі виконується прописана послідовність дій.

Оскільки розумний контракт є відкритим програмним кодом, що може бути написаний програмістом із достатнім рівнем кваліфікації, стає доступним широкий діапазон сфер застосування протоколу.

Широкого застосування розумні контракти здобули в ініціюючих інвестиціях для створення та розвитку перспективних проектів (стартапів). ICO (initial coin offering) передбачає грошовий вклад збоку зацікавлених осіб та організацій існуючими віртуальними валютами, що спрямовано на реалізацію певної ідеї. Оскільки всі транзакції та їх суми запам'ятовуються мережею назавжди, автори стартапу більше не мають потреби вести записи щодо відсотку кожного вкладника. Після успішного запуску проекту, інвестори отримують певну суму нової криптовалюти, що автоматично обраховується пропорційно до його початкового внеску. Вигідність такого вкладу оцінюється ростом вартості до інших відомих валют.

На підставі захищеності даних мережі, відкритості коду зареєстрованого смарт-контракту та неможливості його редагування виникає потенціал для укладання угод без нотаріальної сторони. Наприклад, угода купівлі-продажу майна може виглядати наступним чином: покупець надсилає необхідну суму монет на адресу контракту, а продавець на цю ж адресу відправляє оцифровані документи. Програма перевіряє валідність документів у відповідності до інструкцій, що в неї закладені програмістом. Коли обидві

сторони виконали свої обов’язки, на рахунок продавця надходять монети криптовалюти, а покупцю передаються права володіння. У тому випадку, коли документи не проходять валідацію, кошти повертаються на рахунок платника, а угода скасовується, і навпаки – якщо рахунок не оплачується протягом зазначеного часу, документи втрачають юридичну силу або просто видаляються [7].

З іншого боку смарт-контракт може бути використаним у побуті в поєднанні з технологіями Інтернету речей. Оскільки криптовалютні транзакції не потребують фізичної присутності людини-користувача для підтвердження дій з активами, їх проведення довіряється програмному коду, тобто розумному контракту. Наприклад, побутові прилади, під’єднані до Інтернету речей, можуть купувати електроенергію автоматично, у той час як генератори так само автоматично можуть її продавати. Тобто, генератор надсилає інформацію про об’єм накопиченої енергії на адресу смарт-контракту, до якого він приєднаний, а деякий прилад за допомогою транзакції в мережі купує необхідну йому кількість від доступної.

Ще одним прикладом застосування смарт-контракту є представлення мережі у вигляді глобальної бази даних. Завдяки децентралізації доступ до інформації можна отримати в будь-якому місці в будь-який час, а отже існує потенціал зберігання та передачі інформації за допомогою внутрішніх протоколів системи. Наприклад, покупець замовляє деякий товар в Інтернет-магазині. Після оплати, яка надходить до смарт-контракту, продавець відсилає цей товар. Нехай, доставка має декілька незалежних етапів: вантажне перевезення від складу до порту, переправлення океаном і вантажне перевезення від порту до місця, вказаного замовником. Оскільки такі сервіси як порти для реєстрації та обліку товарів, що проходять через них, працюють із внутрішніми базами даних виникає необхідність використання централізованої системи збору та відображення інформації щодо поточного місця знаходження об’єкту. У такому випадку вирішенням може стати

реєстрація отримання товару на кожному проміжному пункті в смарт-контракті, на який замовник сплатив вартість покупки. Тобто, коли вантажівка прибуває до порту, перевізник відправляє відомість про цю подію до протоколу, а приймаюча сторона підтверджує отримання власним повідомленням. Таким чином, коли перевізник доставляє товар замовнику, він це підтверджує відповідною транзакцією до смарт-контракту. На підставі останнього підпису криптовалюта перераховується з протоколу на рахунок продавця. Слід зазначити, що транзакції про підтвердження доставки-отримання можуть надсилатися так само автоматично, що повністю виключає людський фактор в інформуванні про місце знаходження товару в конкретний момент часу [8].

1.3. Огляд мережі криптовалюти Ethereum

1.3.1. Історія становлення мережі

Ethereum – глобальна платформа з відкритим кодом для розміщення децентралізованих додатків.

Ідея Ethereum'у була описана в 2013 році в статті журналу Bitcoin Magazine Віталієм Бутериним. У своєму виданні автор розповів про потенційні можливості технології блокчейн, які не були реалізовані в уже на той час відомій криптовалюті Bitcoin. Через два роки проект почав свою роботу і одразу привернув увагу банків з метою вивчення технології смарт-контрактів, яка стала основою нової системи.

На даний момент Ethereum – друга за популярністю мережа, тобто її капіталізація (об'єм грошових одиниць в системі) друга за величиною серед інших криптовалют. Вартість (рис 1.1 [9]) не є рекордною, оскільки на це вплинула історія розвитку, а також критичні помилки допущені авторами користувацьких розумних контрактів.

Рис 1.1 Вартість Ethereum протягом усього існування

Найбільше авторитет мережі похитнув провал перспективного проекту The DAO. Стартап на ICO зібрав монет Ethereum'у на суму, оцінену в більш ніж 107 мільйонів доларів США. Причиною краху стала неуважність програмістів, які залишили критично уразливі зони в коді. Результатом ігнорування попереджень збоку експертів став успіх хакерських атак і втрати майже половини інвестованого капіталу, що одразу відзначилося на вартості монет криптовалюти. Для подолання даної проблеми було вирішено відкотити мережу в стан до запуску уразливих смарт-контрактів. Таке рішення знайшло супротивників, які не підтримали цю дію, а продовжили працювати в поточній версії системи, у якій відбувся злам. Ця мережа отримала назву Ethereum Classic і працює до сьогодні.

Головне призначення Ethereum – це платформа для розумних контрактів. Програма може бути спроектована будь-яким програмістом із достатнім рівнем кваліфікації. Розробники криптовалюти пропонують одразу декілька мов на вибір, які можуть використовуватися при написанні протоколу: Solidity (найбільш популярна) та Vyper (орієнтована на безпеку). Також існують такі менш популярні мови як Serpent, LLL і Mutan.

Ідея Ethereum'у полягає у тому, що розвиток мережі відбувається не лише за рахунок удосконалення її технічних аспектів і алгоритмів роботи, а і в утворенні сервісів, що працюють на базі блокчейну криптовалюти.

Реєстрація нових більш досконалих протоколів є поступовою еволюцією екосистеми в цілому.

Завдяки гнучкості смарт-контрактів мережа криптовалюти Ethereum стала найбільш популярною платформою для проведення ICO. Тобто, автори стартапу створюють відповідну програму та реєструють її в системі. Після підтвердження протокол приймає оплату в двох валютах: ETH (Ethereum), BTC (Bitcoin). Натомість інвестор отримує токени стандарту ERC-20 (здебільшого) або ERC-721 (рідше). ERC-20 встановлює певну структуру та ряд правил для розробників, за якими має працювати їхній смарт-контракт, для ERC-721 система більш гнучка і дозволяє реалізовувати власні ідеї з більшим ступенем свободи. Токени можуть виступати акціями проекту, сертифікатами на володіння певними активами, балами в програмах лояльності, криптовалютними активами або виконувати декілька з цих ролей одночасно [10].

Серед інших існуючих проектів із використанням технології смарт-контрактів можна виділити uPort – сервіс для реєстрації в мережі Ethereum. Його ідея полягає у тому, що користувачі не повністю анонімні, а можуть ідентифікувати один одного, обмінюватися інформацією або монетами валюти без загрози несанкціонованого втручання до персональних даних або їх втрати. uPort є прикладом того, як може бути організована виборча кампанія майбутнього з унеможливленням фальсифікацій під час процесу голосування та після його завершення [11].

Одним з найбільш успішних проектів у на базі системи Ethereum є децентралізований додаток другого типу Golem. Сума його зборів ICO станом на листопад 2016-го року оцінюється в 8,6 мільйона доларів США. Сервіс пропонує передавати в оренду обчислювальну потужність комп’ютера через мережу криптовалюти, в якій знаходяться користувачі. Такий самий підхід можна застосувати для оренди вільного дискового простору на віддалених пристроях, що пропонується окремою криптовалютою Filecoin [12].

1.3.2. Структура ключових елементів мережі

Ethereum – це мережа, що має власну криптовалюту, назва якої Ether (у перекладі з англійської – ефір) [13]. Окрім цієї основної грошової одиниці, в системі поширені декілька допоміжних, які є дробовими частинами головної:

– 1 wei = 10^{-18} ETH – найменша одиниця системи;

– 1 gwei (gigawei) = 10^{-9} ETH – основна одиниця оплати комісії.

Мережа Ethereum складається з вузлів, так званих нод (node). Вузол являє собою будь-який комп'ютер підключений до блокчейну криптовалюти. До складу ноди входить чотири основні модулі:

1. P2P Networking Module – модуль для під'єднання до мережі та роботи з іншими її вузлами;
2. Blockchain Module – модуль для організації роботи з блоками блокчейном мережі;
3. Ethereum Virtual Machine – модуль для обробки байт-коду транзакції. Обробка завершується зміною станів акаунтів мережі, що пов'язані даною транзакцією;
4. Account State Database – модуль для зберігання станів акаунтів, що змінюються після успішного підтвердження транзакції. Оскільки всі вузли мережі блокчейну працюють однаково, то і всі бази даних мають однакові записи.

Також існує варіант полегшених (неповних) нод. Такі вузли не зберігають усі дані блокчейну, а лише заголовки блоків для підтвердження дійсності транзакцій, що містяться в цих блоках.

Взаємодія між адресами в мережі криптовалюти відбувається за допомогою транзакцій. Кожна така транзакція має чітко визначену структуру, до її складу входить наступний список полів:

1. nonce – порядковий номер транзакції відносно акаунта, з якого її відправлено;

2. gasPrice – вартість, за якою основні одиниці (ETH) конвертуються в gas (комісійні за виконання транзакції). Оскільки ціна досить невелика, її позначають як GWEI (див. вище);

3. gasLimit – максимальна кількість комісійних, яку готовий сплатити ініціатор транзакції. Тобто, максимальна сума комісії розраховується за формулою gasPrice * gasLimit;

4. to – адреса в блокчейні, на яку відправлені активи;

5. value – сума надісланих активів;

6. v, r, s – використовуються для генерації підпису, що ідентифікує ініціатора транзакції;

7. init – використовується лише для створення нового рахунку контракту, після чого ігнорується;

8. data – необов'язкове для заповнення поле, воно може містити будь-які користувацькі дані [14].

Транзакції групуються в блоки. Окрім певної кількості грошових переказів, блок містить в собі заголовок та набір заголовків інших блоків для оммерів поточного блоку.

Оммер являє собою блок, чий батьківський блок тотожній до прабатьківського блоку поточного. Ця технологія використовується з метою мінімізації втрат таких блоків, що були готові для включення в мережу, однак їх витіснив деякий конкуруючий.

Заголовок блоку у своєму тілі має:

1. parentHash – хеш-вказівник на попередній (батьківський блок). Таким чином усі блоки мережі пов'язані в єдиний ланцюг (звідси походить назва blockchain, де block – блок, chain – ланцюг);

2. ommersHash – хеш списку оммерів поточного блоку;

3. beneficiary – адреса рахунку, яка отримує комісію за обчислення (майнінг) даного блоку;

4. stateRoot – хеш корінного вузла дерева станів;

5. transactionRoot – хеш корінного вузла дерева, що містить усі транзакції даного блоку;

6. recieptsRoot – хеш корінного вузла дерева, що містить квитанції всіх транзакцій даного блоку;

7. logsBloom – структура даних, що складається з логу інформації. Лог складається з адреси регістратора логу, ряду тем, пов'язаних із виконанням транзакцій та будь-які дані, пов'язані з цими задачами;

8. difficulty – рівень складності даного блоку;

9. number – порядковий номер поточного блоку;

10. gasLimit – максимальна кількість комісійних, що може бути нарахована для даного блоку;

11. gasUsed – фактична кількість комісійних нарахована для даного блоку;

12. timestamp – час створення даного блоку;

13. extraData – додаткова інформація щодо даного блоку;

14. mixHash, nonce – хеші, що у поєднанні підтверджують достатність операцій, виконаних для обрахунку даного блоку [15].

Мережа криптовалюти складається з достатньо великої кількості вузлів, що оброблюють транзакції та поміщають їх у блоки. Усі ці дії багаторазово виконуються над одними й тими ж даними кожною з нод, однак абсолютно асинхронно. Це призводить до того, що різні вузли можуть сформувати різні блоки з різними транзакціями. Тобто, наявність блоку в окремій ноді не означає його дійсність – необхідно його узгодити з іншими нодами системи. З метою досягнення консенсусу, тобто обрання єдиного "правильного" блоку, а також вузла, який отримає комісію за виконані обчислення, виникає необхідність визначення певної технології-арбітру.

Таким арбітром в мережі Ethereum на даний момент є алгоритм Proof-of-Work. В своїй роботі, він вимагає внесення від кожного вузла мережі даних

про власну потужність. На основі відсотку обчислювальної потужності конкретної ноди від загальної потужності мережі визначається ймовірність, з якою ця нода отримає комісію за свою роботу, а її блок буде зареєстровано як єдиний правильний.

Особливістю алгоритму Proof-of-Work є використання асиметрії часу виконання трудомісткої роботи, результатом якої є певне значення, та часу на перевірку дійсності отриманого значення. Ця технологія є достатньо стійкою до зламу, однак її використання потребує великої обчислювальної потужності, що тягне за собою великий об'єм споживання електроенергії.

Оскільки складність обчислень лише зростає, Proof-of-Work стає все менш ефективним. При середньому-високому рівні завантаженості мережі середній час генерації та підтвердження блоку складає 15 секунд. Це стало причиною появи ідеї щодо зміни алгоритму консенсусу в блокчейн-системі Ethereum на Proof-of-Stake.

Перевагою Proof-of-Stake є те, що він опирається на пропорції внесених застав для захисту мережі. Така модель значно знижує обчислювальну складність, а отже її використання збільшить потенційну швидкість утворення нових блоків, а тому, ймовірно, приверне увагу вузлів із невеликою потужністю [16].

1.4. Види взаємодії із мережею криптовалюти Ethereum

Щоб стати учасником мережі Ethereum і отримати змогу користуватися внутрішньою криптовалютою необхідно створити власний акаунт в системі (пара відкритого та закритого ключів). Зареєструватися можна за допомогою одного з доступних сервісів, що взаємодіють з блокчейном мережі. Серед таких засобів найбільшу популярність здобули електронні гаманці (wallet) з веб-інтерфейсом, а тому їх вибір є найширшим. Також гаманці існують у вигляді встановлюваного програмного забезпечення для різних операційних систем та у вигляді окремого апаратного забезпечення, однак вони

користуються меншим попитом, оскільки орієнтовані на більш досвідчених користувачів та потребують підвищеного рівня уваги до компонентів комп'ютера, як збоку продуктивності, так і захищеності. Основна задача таких додатків – зберігати приватні ключі акаунтів користувача.

Функціонал гаманців не обмежується проведенням транзакцій в мережі криптовалюти. Він також дозволяє керувати зареєстрованими акаунтами (можна додати декілька власних адрес), переглядати історію грошових переказів, дані щодо вартості та капіталізації криптовалют у реальному часі.

Для сумісного користування певним рахунком декількома користувачами можна скористатися смарт-контрактом. Акаунти, які мають право власності на протокол вказані в самому протоколі. Здебільшого для перерахування коштів з таких смарт-контрактів необхідно отримати підтвердження від усіх (або принаймні більшості) власників у вигляді транзакції-підпису. Дана технологія називається мультипідписом (multisig). Така взаємодія поширена в протоколах, що формуються для ICO.

Для зручності взаємодії існують multisig-гаманці. Вони складаються з контракту та графічного інтерфейсу. Подібно до того, як звичайні гаманці можуть зберігати одразу декілька акаунтів, multisig-гаманці можуть працювати з кількома контрактами водночас, однак механізм роботи мають інший і тому прирівнювати сервіси не можна [18].

Оскільки загальна інформація блокчейн мереж знаходиться в публічному доступі, для взаємодії з нею реєстрація акаунту не є обов'язковою. Тобто, можна не брати участі в існуванні системи, а лише стежити за транзакціями адрес криптовалюти, їхніми поточними балансами та базовими даними, що знаходяться в мережі.

Також використання цього підходу є зручним, оскільки для перегляду необхідної інформації немає потреби в аутентифікації, як це відбувається при роботі з електронним гаманцем. Тобто, користувач може зареєструватися в мережі за допомогою одного з сервісів, а отримувати інформацію щодо участі

свого акаунта в транзакціях з відкритих джерел. Це значно швидше та безпечніше, аніж кожного разу вводити приватні дані при вході до власного гаманця.

Для сервісів, що не потребують реєстрації залишається відкритою можливість здійснення криптовалютних транзакцій. Реалізація полягає у використанні спеціального API мережі криптовалюти. Кожен грошовий переказ потребує підпису приватним ключем акаунта. Тобто, знаючи даний ключ, платник не має необхідності в користуванні електронним гаманцем, а може кожну свою транзакцію підписувати власноруч (якщо сервіс пропонує даний функціонал).

ВИСНОВОК ДО РОЗДІЛУ 1

У даному розділі розглянуто предметну область дипломного проекту, її особливості становлення, основні поняття, принципи роботи, переваги та недоліки і перспективи розвитку в майбутньому.

Проведено детальний огляд цільової системи, структури основоположних елементів мережі – її вузлів. Проаналізовано структуру даних та самі дані, їхнє походження та призначення, якими оперує програмна сторона системи. Також розглянуто різні принципи взаємодії з мережею та способи їхньої реалізації.

РОЗДІЛ 2

ФОРМУЛЮВАННЯ ФУНКЦІОНАЛУ ТА АНАЛІЗ ТЕХНОЛОГІЙ

2.1. Постановка задачі

2.1.1. Призначення розробки

Розроблюваний проект призначений для відображення транзакцій в мережі криптовалюти Ethereum, відфільтрованих за певною адресою. Користувачем програми може бути як зареєстрований учасник системи, так і користувач без власного акаунту.

2.1.2. Задачі та мета розробки

Під час реалізації мають бути вирішені наступні задачі:

– Мінімізувати вимоги до апаратного та програмного забезпечення комп’ютерної системи;

– Досягнути конкурентоспроможної швидкості оновлення даних;

– Створити зручний інтуїтивно зрозумілий інтерфейс для користувача;

– Мінімізувати та структурувати об’єм збережених користувацьких даних;

– Надати користувачу достатній об’єм функціональних можливостей.

Метою розробки є створення сервісу сповіщення про нові транзакції обраного акаунту в мережі криптовалюти з високою пропускною здатністю та середнім-високим рівнем стійкості до навантаження за умови обмеженості апаратного забезпечення.

2.1.3. Опис функціональної моделі

Розроблюваний проект передбачає взаємодію користувача з мережею криптовалюти Ethereum через власний інтерфейс. Тобто, сервіс має одну роль – користувач (особа, зацікавлена в процесах життєдіяльності певних адрес цільового блокчейну).

Таким чином в проекті використовується технологія схожа до оберненої дошки оголошень. Тобто, користувач заявляє про свої потреби, а обслуговувач (у даному випадку – програма) у відповідності до своїх можливостей виконує замовлення або сповіщає про неможливість його виконання (наприклад, необхідна адреса не відноситься до адресного простору мережі криптовалюти Ethereum).

2.1.4. Функціонал розробки

Розроблюваний проект має автоматизувати наступні процеси:

1. Пошук адреси в мережі криптовалюти;
2. Пошук здійснюваних транзакцій адреси мережі криптовалюти;
3. Моніторинг здійснення нових транзакцій адреси мережі криптовалюти.

З метою реалізації запланованого проекту має бути розроблено наступні функціональні можливості:

1. Обробка запиту на додавання до програми акаунту мережі криптовалюти для відстежування;
2. Перевірка дійсності наданої адреси мережі криптовалюти;
3. Підтримка користувацьких символьних імен для доданих адрес блокчейну;
4. Сповіщення про нові транзакції за участі відстежуваного акаунта у вигляді push-повідомлень;

5. Регульована фільтрація сповіщень про виявлення нових транзакцій обраного акаунта;

6. Відображення історії здійснюваних транзакцій обраної адреси мережі криптовалюти;

7. Регульована фільтрація записів історії здійснюваних транзакцій обраної адреси блокчейну.

2.1.5. Проектування розробки

Розроблювана система є посередником між цільовим користувачем та мережею криптовалюти. З метою забезпечення роботи кінечних автоматів у процесі взаємодії з інтерфейсом програми також має бути застосовано модуль для зберігання користувацьких даних. Отже, розроблювана система буде складатися з трьох функціональних елементів:

1. Інтерфейс користувача;
2. Інтерфейс локальної бази даних;
3. Інтерфейс роботи з мережею криптовалюти.

Взаємодію елементів та інтерфейсів розроблюваної системи схематично представлено на рисунку 2.1.

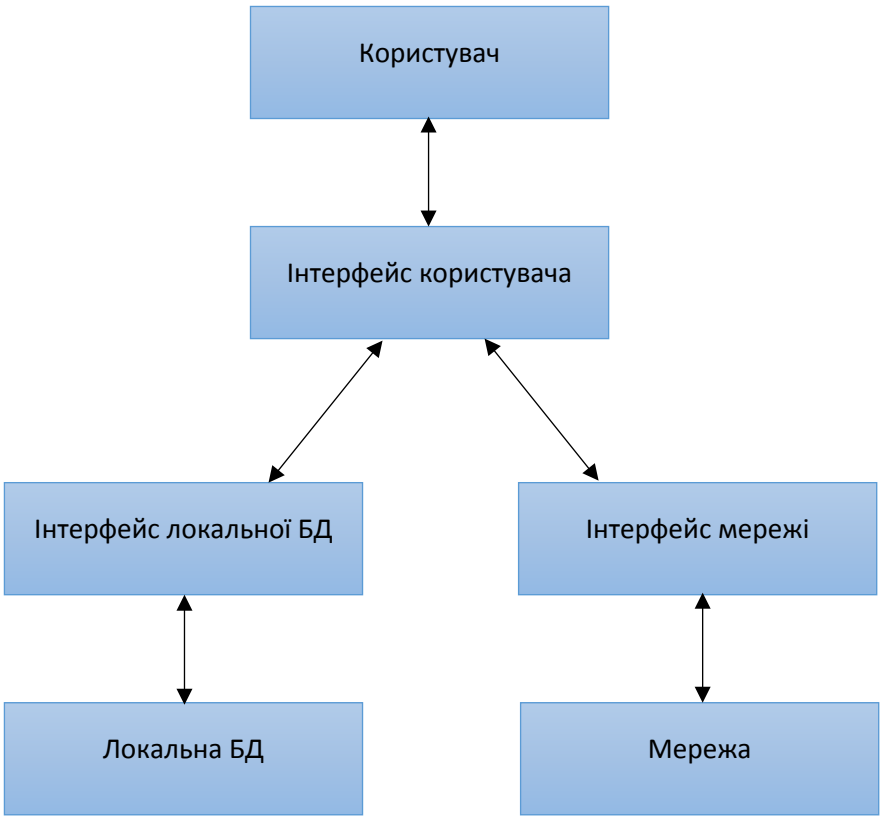


Рис. 2.1. Схема взаємодії користувача та розроблюваного проекту

Взаємодія користувача та сервісу, починається з привітання та пропозиції ввести адресу мережі криптовалюти для відстежування збоку програми. Після реєстрації акаунтів зацікавлена особа може керувати повідомленнями про нові транзакції (вмикати та вимикати). Також додана адреса може бути видалена за бажанням користувача.

2.2. Короткий огляд альтернативних мереж

Ідея смарт-контрактів в Ethereum стала досить популярною, що спричинило утворення альтернативних мереж, які використовують дану технологію. Серед них можна виділити наступні:

1. NEO – проект, ціллю якого є розвиток “розумної економіки”. За різними оцінками, перевагою цієї криптовалюти перед Ethereum є більш продумана робота зі смарт-контрактами: віртуальні машини мережі можуть оптимізувати код програми перед його виконанням,

однак водночас це є недолік, оскільки сповільнюється запуск, а отже і обробка блоку в цілому;

2. Nxt – платформа для запуску децентралізованих додатків з відкритим кодом. Вона має обмежений набір шаблонів для розумних контрактів, однак написання власних не дозволено;

3. Jincor – платформа, головною специфікацією якої є бізнес. Даний проект дозволяє створити смарт-контракт за допомогою спеціального конструктора або скористатися одним з уже існуючих, а отже для підприємців зникає потреба в кваліфікованому програмісті;

4. Qtum – гібридна платформа, націлена на взаємодію із бізнесом за допомогою так званих мастер-контрактів. Від смарт-контрактів такі протоколи відрізняються тим, що підписання та розрив договорів перекладається на самих учасників договору. Ще одна особливість полягає у тому, що криптовалюта платформи сумісна з монетами Bitcoin та Ethereum;

5. Ubiq – проект, побудований на технології блокчейну Ethereum. Платформа орієнтована на підтримку автоматизованих смарт-контрактів із високою пропускною здатністю [17].

У порівнянні з вище наведеними альтернативними мережами криптовалют, Ethereum має наступні переваги:

- Найбільш популярна криптовалюта;
- Найбільш розвинута мережа;
- Гнучка система роботи зі смарт-контрактами;
- Різноманітна сфера застосувань.

Отже, можемо зробити висновок, що мережа криптовалюти Ethereum потенційно є найбільш перспективною, оскільки вона має ґрунтовні можливості для подальшого розвитку, який не обмежується конкретним напрямком чи певною специфікацією.

2.3. Огляд наявних аналогів у цільовій галузі

Серед сервісів, що не потребують реєстрації, але дають можливість перегляду інформації мережі криптовалюти Ethereum можна виділити веб-портал Etherscan (найпопулярніший серед подібних) та чат-бот на базі технологій месенджера Telegram ETHtokenExplorer_bot.

Таблиця 2.1 Порівняння аналогів (початок)

Характеристика	**Etherscan**	**ETHtokenExplorer_bot**
1	**2**	**3**
Тип інтерфейсу	Веб-сторінка	Чат-бот
Швидкість відображення нових даних	Майже миттєво	Затримка від 30 до 60 секунд
Спосіб взаємодії із мережею	Базується на роботі вузла мережі	Базується на роботі повного вузла мережі
Спосіб відображення даних	Відображення інформації з індексованої локальної бази даних з використанням мережевого протоколу RPC для зменшення часу звернення до сховища та очікування відповіді	Відображення інформації з локальної бази даних через прямі звернення до неї
Можливість фільтрації даних	Так, необов’язково	Так, обов’язково з додатковими необов’язковими параметрами

Таблиця 2.1 Порівняння аналогів (закінчення)

1	2	3
Відображувана інформація	Відображено усі блоки мережі та транзакції, що входять до їх складу з усіма даними, що з ними пов’язані, а також інформація про поточну вартість та капіталізацію валюти, вартість та капіталізацію токенів, стани розумних контрактів, рейтинг майнерів, кількість транзакцій і сумарну складність обрахунків в режимі реального часу, базові теоретичні відомості про мережу та її складові	Відображена інформація щодо зареєстрованих на відстежування акаунтів (до двох адрес одночасно): їхній баланс, нові та останні транзакції
Додаткові можливості	Верифікація смарт-контракту, декодування байт-коду транзакцій, компілятор для мови Vyper, калькулятор майнінгу, публічний API	Автоматична конвертація валют
Наявність push-повідомлень	Ні	Так

З порівняльної таблиці видно, що сервіс Etherscan надає значно глибші відомості про поточний стан мережі, а також має широкий вибір

функціональних додатків. ETHtokenExplorer_bot орієнтований на більш обмежений інтерес користувача. Тобто, чат-бот надає вже відфільтровану від надлишкових даних інформацію, у той час як веб-портал потребує відповідних дій збоку зацікавленої особи для відображення цільових транзакцій. Також знаковою відмінністю сервісів є наявність push-повідомлень у ETHtokenExplorer_bot, що виключає необхідність періодичного опиту мережі збоку користувача. Серед спільного в обох проектах можна виділити використання власного вузла мережі, що потребує технічного та програмного забезпечення підвищеного рівня.

2.4. Аналіз технологій взаємодії з мережею криптовалюти

Найбільш поширеним способом взаємодії з блокчейном деякої системи є налаштування власного вузла мережі. У такому випадку розробнику необхідно мати окремий комп'ютер-сервер, на який встановлюється відповідне програмне забезпечення. Від працюючої ноди за допомогою будь-якого з доступних методів відбувається передача інформації, записаної в локальну базу даних вузла мережі (вона знаходиться на сервері), до персонального комп'ютера зацікавленого користувача.

Такий підхід у середньому потребує великого об'єму пам'яті твердотільного запам'ятовуючого пристрою (не менше одного терабайту) виконаного за технологією SSD, чотирьох ядерного процесора та восьми гігабайтного оперативного запам'ятовуючого пристрою. Також обов'язковою є підтримка неперервного з'єднання з мережею Інтернет, оскільки з її допомогою вузли мережі обмінюються інформацією.

Альтернативним рішенням проблеми отримування даних з мережі криптовалюти є використання публічних API. Такий підхід не потребує від розробника-користувача високопродуктивної комп'ютерної системи або великого об'єму пам'яті. Уся інформація отримується лише за певним запитом

до дистриб'ютора. Відповідно, даний підхід з апаратної частини потребує лише стабільного зв'язку з мережею Інтернет.

Для досягнення поставленої мети обирається другий варіант взаємодії з мережею криптовалюти Ethereum.

2.5. Короткий огляд дистриб'юторів API

Публічний API надається багатьма сервісами в цільовій сфері. Наприклад, на вище розглянутому популярному порталі Etherscan можна знайти всі методи, необхідні для досягнення мети проекту. Після проходження реєстрації на сайті користувачу надається можливість безкоштовно отримати до двох ключів, які дозволяють користуватися публічним API. Однак, Etherscan накладає певні обмеження на користування власним інтерфейсом, серед яких максимальна кількість запитів на секунду збоку стороннього сервісу, що не має перевищувати п'яти звертань.

Подібний функціонал надають реалізації бібліотеки web3 для різних мов програмування (серед найпопулярніших – Python та JavaScript). Для роботи з ними також необхідно додатково вказати адресу цільової мережі (провайдера). У такий спосіб розробник отримує можливість налагодити роботу власної програми в локальній (тестовій) мережі перед запуском готового проекту в повноцінному середовищі.

Однак, вище зазначені сервіси працюють за принципами протоколу передачі даних HTTP. Тобто, схема роботи з ними виглядає як обов'язковий ланцюжок дій "запит" – "відповідь". Оскільки формування кожного нового блоку займає різний проміжок часу, робота за наведеною схемою призведе або до затримок обробки нової інформації та нерівномірної завантаженості серверу (за умови проведення опитування не частіше одного разу на кілька секунд), або до перенавантаженості серверу та блокування збоку дистриб'ютора (при опитуванні частіше одного разу на секунду). Отже, для

досягнення поставленої мети у розроблюваному проекті використання методів наведених відкритих API є недоцільним.

Для вирішення поставленої задачі ефективним є застосування протоколу WebSocket. Його робота, так само як і при використанні методів HTTP, починається з клієнтського handshake-запиту до дистриб'ютора, однак особливість у тому, що після успішного рукопотискання клієнт більше не звертається до API з метою отримання інформації, а сервіс сам надсилає необхідні дані, щойно їх отримує.

На основі даного протоколу працюють деякі методи відкритого API, запровадженого веб-порталом INFURA. Даний сервіс так само, як і Etherscan потребує реєстрації та отримання власного ключа, а натомість він надає широкий вибір методів взаємодії з мережею, схожих до методів у бібліотеці web3, у тому числі наявна можливість вибору провайдера.

Отже, у якості дистриб'ютора API мережі криптовалюти Ethereum обрано публічний сервіс INFURA.

2.6. Огляд способів реалізації інтерфейсу користувача

Оскільки метою проекту є доставка користувачу миттєвих push-повідомлень про транзакції відстежуваних акаунтів, найбільш доцільним є запровадження інтерфейсу сумісного з мобільними пристроями. Подібні девайси тримаються увімкненими протягом усього часу, коли користувач готовий до взаємодії з інформацією, що надходить з мережі. Отже, розглянемо наявні варіанти рішення даного питання.

Очевидним підходом є створення власних додатків для мобільних платформ. Перевагою такого рішення є можливість повного візуального та функціонального налаштування інтерфейсу програми. Також, оскільки користувацький додаток необхідний лише для відображення інформації та не потребує виконання складних обрахунків, його реалізація не потребує

великого об’єму пам’яті та може бути сумісною із більш застарілими версіями мобільних платформ, а отже і з непотужними пристроями.

Серед недоліків такого підходу можна виділити необхідність його окремого встановлення на портативний комп’ютер. Для користувача це означає невелику, однак втрату об’єму вільної пам’яті. На даний момент багато виробників мобільних пристроїв не додають можливість розширити внутрішній накопичувач зовнішніми модулями, а тому питання експлуатації запам’ятовуючого апаратного забезпечення може стати вирішальним для користувача.

Ще одним недоліком створення власного програмного інтерфейсу користувача є необхідність його виконання для різних платформ. Щонайменше має бути створено додатки для двох найпопулярніших операційних систем мобільних пристроїв: Android та iOS.

Альтернативним рішенням є використання наявних засобів із можливістю надсилання сповіщень користувачу. Найбільш ефективно функціонал push-повідомлень реалізовано в месенджерах – програмах, призначених для обміну миттєвими повідомленнями. Серед таких сервісів найбільш популярними в Україні за статистикою, проведеною у 2019 році є Viber (використовується на 96,2% мобільних девайсів країни), Facebook Messenger (67,1%) і Telegram (46,7%) [19].

Не зважаючи на великий розрив у відсотку пристроїв, що активно використовують перший та третій месенджер рейтингу, Telegram вважається найбільш перспективним завдяки потенціалу технології чат-ботів, що пропонує платформа. Ідея віртуального співрозмовника була розвинута до утворення гібриду між звичними принципами побудови інтерфейсу програми та веденням діалогу в соціальній мережі.

Відповідно, Telegram пропонує власні додатки для всіх популярних операційних систем. Серед підтримуваних платформ наявні не лише цільові

рішення для мобільних пристроїв, але й для десктопних операційних систем у тому числі.

Для розробників чат-ботів Telegram пропонує відкритий глибоко документований Telegram Bot API, що підтримується та постійно удосконалюється програмістами сервісу. Також месенджер надає власні реалізації бібліотек для популярних мов, які виконують звернення до методів офіційного інтерфейсу.

На даний момент існує велика кількість розроблених чат-ботів, що надають найрізноманітніший функціонал. Такі програми можуть допомагати у виконанні рутинних справ, задовольняти розважальні потреби або використовуватись як інтерфейс Інтернет-магазину.

Одними з найбільш відомих переваг розглянутого месенджеру є його захищеність та відносна анонімність. Для обміну повідомленнями опціонально застосовується наскрізне шифрування, що убезпечує діалог від компрометації через злам віддаленого серверу. Оскільки більшість користувачів криптовалютних мереж переслідують саме вище наведені особливості, доєднуючись до блокчейн-систем, Telegram став досить популярним рішенням в їхній спільноті.

Отже, на підставі вище наведених переваг, для реалізації інтерфейсу користувача створюваного проекту обрано месенджер Telegram та його технологію віртуальних співрозмовників.

2.7. Вибір мови програмування

Для взаємодії з інтерфейсом блокчейну мережі криптовалюти Ethereum найчастіше застосовується динамічна мова програмування JavaScript. Вона здобула широкої популярності завдяки своїй зручності у написанні додатків для мережі Інтернет, зокрема веб-сервісів.

Для даної мови програмування створена досить велика кількість сторонніх бібліотек, серед яких вище розглянута web3 – для роботи з мережею

цільової криптовалюти та node-telegram-bot-api – для розробки чат-ботів на платформі месенджеру Telegram.

До переваг JavaScript також можна віднести підтримку асинхронного стилю розробки. Такий підхід до програмування був популярний до розвитку багатопотчності. Він покликаний створювати видимість так званої мультизадачності – одночасного виконання декількох додатків. Тобто, асинхронність за рахунок швидкої зміни контексту ядра процесора дозволяє частково обробити одну з готових до виконання задач (задача, що знаходиться в черзі RunQueue), потім, у залежності від потреби, або повернути її до тієї ж черги з меншим пріоритетом, або покласти до іншої. Натомість обирається наступна задача з RunQueue так само частково обробляється, і так само повертається в необхідну чергу. Такі дії повторюються за принципами планування Round-robin до повного виконання задач.

У зв'язку зі збільшенням навантаження на веб-сервіси асинхронний стиль повернув собі популярність. Оскільки його використання доцільне для вирішення задач пов'язаних із очікуванням вводу-виводу, що є ключовим для додатків працюючих із мережею Інтернет, підтримка цього підходу дала JavaScript сильний поштовх у розширенні аудиторії.

Альтернативою у даному виборі є не менш популярна мова програмування Python. Вона відома лаконічністю та зрозумілістю свого синтаксису, що дозволяє створювати робочі проекти за значно менший проміжок часу, аніж більш потужні Java, C# або C++.

Python – високорівнева скриптова мова програмування загального призначення. Розгляд коду з точки зору сценарію робить його зручним для читання, редагування та розширення. Завдяки вбудованій обробці помилок відлагодження програми потребує від розробника менших зусиль та займає значно менше часу, аніж на C-подібних мовах.

Python написано на всесвітньо відомій мові програмування C. Остання завдяки своїй потужності здобула широкої популярності у створенні

низькорівневих програм, у тому числі мережевого забезпечення. Хоча Python не успадкувала від свого “батька” швидкості роботи, сумісність із C дала поштовх для застосування мови в мережевих технологіях. Існує чимало як стандартних (тобто таких, що поставляються разом з інтерпретатором), так і користувацьких бібліотек для роботи з Інтернет-протоколами на Python.

Так само як і JavaScript, Python підтримує асинхронний стиль програмування. Не зважаючи на те, що повноцінна підтримка конкурентності з’явилася не так давно, чимала кількість розробників уже звернула увагу на цей напрям розвитку та додала до власних програм і бібліотек сумісність з нововведеною технологією.

Python є одним із лідерів рейтингу мов програмування за кількістю написаних сторонніх бібліотек. Серед них можна знайти різні реалізації методів для роботи з, ймовірно, будь-якою базою даних, взаємодії з API чат-ботів месенджеру Telegram чи інтерфейсом блокчейну мережі криптовалюти Ethereum (у тому числі web3). Відкритість коду кожної бібліотеки дозволяє програмісту вільно його редагувати чи модифікувати під власні потреби, однак така необхідність виникає не часто.

Оскільки дана проектна робота не ставить задачі побудувати повноцінний веб-сервіс, для досягнення мети більш доцільно обрати альтернативну мову програмування Python.

2.7.1. Короткий огляд засобів створення інтерфейсу

Для взаємодії з API месенджеру Telegram Python пропонує декілька сторонніх бібліотек. Серед них можна виділити дві найпопулярніші:

– pyTelegramBotAPI – зручна у використанні бібліотека, що підтримує синхронний стиль програмування. Вона використовує іншу сторонню синхронну бібліотеку для роботи із запитами за протоколом HTTP – requests;

– aiogram – більш нова бібліотека, що успадкувала принципи побудови програми від pyTelegramBotAPI, однак модуль requests було замінено асинхронним aiohttp. Це дало змогу побудувати більш потужний та стійкий до навантажень інтерфейс, який також підтримує вбудовані методи взаємодії з базою даних.

Відповідно, для реалізації користувацького інтерфейсу проекту обрано другу бібліотеку.

2.8. Вибір засобів зберігання та передачі інформації

Оскільки програма передбачає роботу з різними типами інформації, з метою реалізації проекту має бути застосовано декілька сховищ даних, що працюють за різними принципами.

2.8.1. Взаємодія користувача та програми

Виключно для роботи користувача з інтерфейсом програми використовуються вбудовані в фреймворк aiogram методи взаємодії з розподіленою NoSQL базою даних Redis. Вона зберігає інформацію в оперативному запам’ятовуючому пристрої, за принципом ключ-значення. Redis працює зі строковими типами даних і підтримує наступні варіанти їхнього застосування у якості значень:

– Рядок – основний тип даних, максимальний доступний розмір 512 мегабайт;

– Список рядків – упорядкований за послідовністю вставки, максимальна довжина 2^{32}-1 елементів;

– Множина рядків – неупорядкована, відповідає математичному змісту терміну, максимальна довжина 2^{32}-1 елементів;

– Хеш-таблиця – масив пар ключ-значення, максимальна довжина 2^{32}-1 елементів;

– Упорядкована множина рядків – відповідно, множина, описана вище, упорядкована за спеціальним параметром “score”.

Оскільки зазвичай програми працюють не лише з рядками, а й іншими типами даних (наприклад, числа), в aiogram передбачено попереднє приведення отримуваної інформації до json-об’єкту, який потім завантажується в строкове представлення для передачі в Redis. Відповідно, на запит користувача, потрібні дані дістаються з бази та вивантажуються назад у json.

Redis також надає можливість проводити відповідні операції зі збереженими даними, однак aiogram передбачає лише додавання, оновлення та видалення пари ключ-значення. З метою використання доступних шляхів взаємодії з інформацією, а також для відлагодження програми розробник може скористатися інтерфейсом Redis в командному рядку (redis-cli), який поставляється разом із базою даних.

Головною перевагою використання Redis є висока швидкість доступу до даних. Однак, за рахунок використання саме оперативної пам’яті, розміщення великого об’єму інформації призведе до використання swap-розділу жорсткого диску, що сповільнить роботу програми. Отже, дана технологія має бути використана для зберігання проміжних даних, що виникають під час взаємодії користувача та інтерфейсу програми.

2.8.2. Взаємодія програми та даних

У якості інтерфейсу для взаємодії з основним сховищем інформації може бути застосована одна з найбільш поширених систем для керування реляційними базами даних – MySQL. Вона проста та зрозуміла у використанні. Головною її перевагою є швидкість роботи, яка не залежить від об’єму збереженої інформації.

MySQL підтримує ряд стандартних типів даних більшості мов програмування, що робить її використання досить зручним, оскільки

програмісту не потрібно продумувати попередню обробку інформації перед її внесенням до сховища і обернену обробку після діставання. Усі наявні типи даних можна поділити на групи:

– Символьні типи (рядки): CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, LARGETEXT;

– Числові типи: TINYINT (BOOL), TINYINT UNSIGNED, SMALLINT, SMALLINT UNSIGNED, MEDIUMINT, MEDIUMINT UNSIGNED, INT, INT UNSIGNED, BIGINT, BIGINT UNSIGNED, DECIMAL (NUMERIC, DEC, FIXED), FLOAT, DOUBLE (REAL, DOUBLE PRECISION);

– Дата-час типи: DATE, TIME, DATETIME, TIMESTAMP, YEAR;

– Складені типи: ENUM, SET;

– Бінарні типи: TINYBLOB, BLOB, MEDIUMBLOB, LARGEBLOB.

У кожній групі є по декілька типів, які здебільшого різняться між собою максимальною доступною величиною, що може зберігатися (відповідно, для них використовується різна кількість пам’яті) та символьним відображенням для користувача.

Для окремої ідентифікації кожного рядка, таблиці бази даних для них обов’язково зберігають унікальні номери або значення (первинні ключі), що мають назву primary key. Зазвичай вони задаються користувачем, однак якщо він цього не робить, ключ створюється автоматично під назвою “id”. На основі технології унікальних полів (індексів) будуються реляційні моделі в MySQL. Наприклад, за її допомогою вирішується відома в побудові структури баз даних задача відношення багатьох до багатьох.

З метою ефективного використання реляційної системи керування базами даних, необхідно притримуватися певних правил розміщення інформації у сховищі. Найпоширенішим методом оптимізації роботи є

нормалізація. Вона допомагає забезпечити захист та гнучкість бази даних за рахунок виключення надлишкових записів та неузгодженості зв’язків. Усього існує шість (ітеративних) нормальних форм, серед яких кожна наступна повністю задовольняє умови попередньої:

1. Усі атрибути є простими, а усі домени містять лише скалярні значення;
2. Кожен неключовий елемент залежить від первинного ключа;
3. Кожен ключовий елемент нетранзитивно залежить від первинного ключа;
4. Усі нетривіальні залежності є функціональними залежностями від потенційних ключів;
5. Відсутні складені з’єднання між атрибутами;
6. Кожна змінна відповідає всім нетривіальним залежностям з’єднання та не може бути декомпонована.

Отже, за умови правильного використання, MySQL є досить ефективним рішенням для задоволення потреб у центральному сховищі для великого об’єму інформації. Дана система надає достатню швидкість взаємодії з інформацією, зі збереженням можливості налаштування середовища, зручності у використанні, гнучкості для розробки та високим рівнем захисту від несанкціонованого втручання збоку зловмисників.

2.8.3. Взаємодія програми та мережі криптовалюти

Оскільки мережа криптовалюти та головний процес розроблюваної програми працюють незалежно один від одного, виникає необхідність синхронізації роботи окремих частин проекту. З метою налаштування взаємодії незалежних процесів ефективним рішенням є організація обміну повідомленнями між цими підпрограмами за допомогою деякого посередника (брокера), що працює незалежно від обох сторін.

Головним незалежним процесом мережі, що підлягає синхронізації, є утворення нових блоків. Тобто, реалізація вимагає розробки двох слабо пов’язаних підпрограм:

1. Утиліта для отримання блоку від мережі;
2. Утиліта для обробки блоку.

Обробка нового блоку включає в себе повний огляд його транзакцій та, якщо є співпадіння, надсилання користувачу повідомлення про нову транзакцію в мережі за участі відстежуваного ним акаунта.

Застосування зазначеної архітектури передбачає також можливість запуску обробника в декількох процесах. Тобто, можлива така ситуація, коли популярність мережі Ethereum значно зросте, а отже нові блоки будуть створюватися з надто великою швидкістю. У такому випадку розроблена програма може не встигати обробляти всі запити. Однак, за рахунок того, що брокер є незалежною стороною та може передавати дані декільком споживачам з однієї черги, кілька обробників, працюючи на різних ядрах комп’ютерної системи будуть забезпечувати користувачів актуальною інформацією значно швидше та ефективніше.

Зв’язок між утилітами доцільно організувати за допомогою системи обміну повідомленнями, яка за сумісністю виконуватиме роль черги, що утримує інформацію перед подальшою обробкою. Вигідним рішенням даного питання є платформа RabbitMQ.

RabbitMQ являє собою сервіс для доставки повідомлень. У своїй роботі він підтримує декілька протоколів передачі даних, найпопулярнішими серед яких є AMQP (використовується за замовчуванням) та HTTP.

Робота протоколу AMQP може бути описана в три етапи:

1. Прийняття повідомлення (exchange). Його часто порівнюють із поштовою скринькою, куди необхідно покласти лист;

2. Направлення повідомлення до потрібної черги (route). Цей етап характеризують як листоношу, що забирає лист із поштової скриньки та відносить до місця призначення;

3. Читання повідомлення із черги (queue). Черга (місце призначення) порівнюється із особистою поштовою скринькою адресата, куди листоноша поклав повідомлення. Відповідно, адресат має зазирнути до власної скриньки та забрати звідти новий лист.

Така технологія є досить простою для розуміння та дуже гнучкою у використанні. Вона дозволяє одночасно працювати із великою кількістю черг, які можуть використовуватися кількома програмами як з метою надсилання інформації, так і її отримування.

Отже, реалізація синхронізації утиліт працює за наступним алгоритмом: інтерфейс взаємодії з блокчейн-системою (publisher) отримує інформацію про створення нового блоку мережі криптовалюти. Він, за допомогою RabbitMQ, має надіслати (publish) цю інформацію обробнику (consumer). Останній повинен прочитати (consume) повідомлення із відповідної черги, повністю переглянути отримані дані та, за умови знайденого співпадіння номерів адрес мережі криптовалюти, надіслати користувачу програми повідомлення про нову транзакцію.

2.9. Середовище розробки

Для розробки проекту обрано UNIX-подібну операційну систему, найбільш популярний дистрибутив Linux – Ubuntu 16.04.6 LTS. Ця версія є найбільш стабільною серед усіх наявних, при чому вона підтримується розробниками до сьогодні, а отже для неї наявне все сучасне програмне забезпечення. Важливою перевагою даної системи є робота з мовою програмування Python за замовчуванням, що виключає необхідність великої кількості налаштувань середовища перед початком роботи. Також, на підставі

стабільності обраної платформи, планується розгортання програми на сервері під керуванням аналогічної операційної системи.

У якості середовища редагування та відлагодження коду було обрано PyCharm за авторством компанії JetBrains. Дане середовище відноситься до більш професіональних рішень, оскільки воно підтримує роботу не лише файлами коду, а й оточенням операційної системи в цілому. Окрім редагування та відлагодження декількох модулів одночасно, користувач має можливість працювати з базою даних, структурою проекту, запускати декілька програм на виконання, взаємодіяти із командним рядком чи інтерпретатором мови в одному вікні програмного засобу. Дане середовище повністю відповідає потребам, що будуть виникати під час розробки додатку.

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі було поставлено мету проекту та задачі, що мають бути вирішені для її досягнення. Описано функціонал розроблюваного додатку. Спроектовано архітектуру майбутньої системи та способи взаємодії її внутрішніх компонентів.

Проведено аналіз технологій, що можуть бути використані при розробці системи. Серед розглянутих рішень було обрано найбільш доцільні для виконання даної роботи.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. Розробка алгоритму обробки нових блоків

Як було зазначено вище, обробка нових блоків мережі криптовалюти має бути розділена на два етапи:

1. Прийом повідомлення про новий блок;
2. Обробка нового блоку.

Взаємодія підпрограм організована за допомогою технології доставки повідомлень незалежного посередника (брокера) – сервісу RabbitMQ. Він також виконує роль черги та невеликого сховища для утримання проміжних тимчасових даних.

3.1.1. Алгоритм прийому блоків

Оскільки перший етап полягає у взаємодії з мережею, він має виконуватися з мінімальним користувацьким навантаженням. Тобто, доцільно організувати його алгоритм роботи якомога більш лінійним та невибагливим до ресурсів, аби підпрограма мала готовність отримати нову порцію інформації заздалегідь. Даний алгоритм зображено на рисунку 3.1.

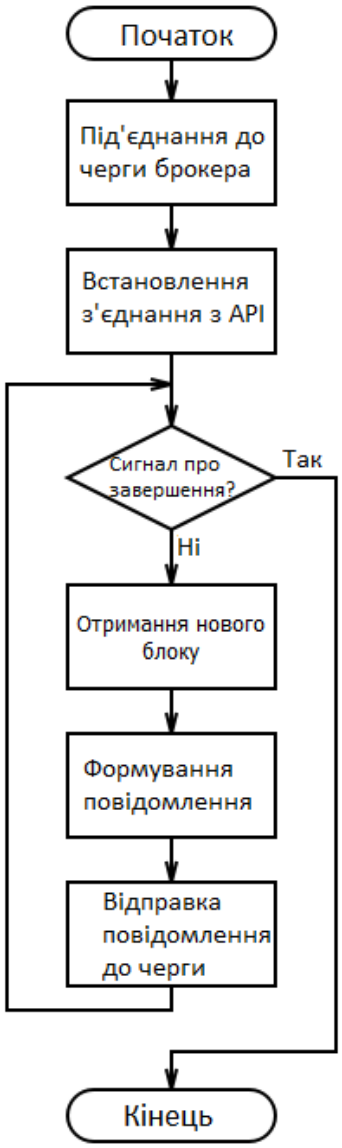


Рис. 3.1. Алгоритм прийому нових блоків

Як видно з рис. 3.1., алгоритм має лише одне розгалуження, що відповідає за цикл, у якому виконується основний код програми. Такий підхід є досить продуктивним, оскільки виконання кожної з дій циклу не потребує великих обчислювальних та часових затрат.

3.1.2. Алгоритм обробки блоків

Другий етап є більш складним і потребує деяких часових та обчислювальних затрат.

Головним чином на час виконання підпрограми впливає особливість використовуваного API: сервіс INFURA надсилає повідомлення про новий

блок, щойно він з’являється в його вузлі. Однак, наявність блоку в окремій ноді не означає його дійсність – необхідно отримати підтвердження реєстрації в мережі алгоритмом консенсусу. Відповідно, дані, що містяться в новому блоці залишаються недоступними деякий час, а отже надісланий запит до API з метою отримання списку транзакцій не дасть результату.

Аби мінімізувати кількість запитів до сервісу, а також зменшити навантаження на власну обчислювальну машину, можна встановити штучну часову затримку. Використання асинхронного підходу до написання програми означає, що даний “простій” є актуальним виключно для конкретного блоку і ніяк не впливає на роботу програми в цілому.

Також є необхідним отримання внутрішніх транзакцій. Такі транзакції мають деякі особливості, а тому поставляються окремо від звичайних. Це означає, що виникає потреба в додатковому зверненні до API, що збільшує часові затрати на обробку блоку.

Однією з особливостей внутрішніх транзакцій є їхнє походження – вони породжуються звичайними транзакціями. Оскільки можлива ситуація, коли в блоці немає жодного грошового переказу, запит щодо внутрішніх транзакцій не завжди є доцільним. Отже, виконуючи таку перевірку можна також зменшити час обробки інформації.

Після об’єднання усіх транзакцій в єдиний список, залишається перевірити кожну адресу, чи відстежується вона хоча б одним користувачем програмного продукту. Якщо користувача знайдено – надіслати йому повідомлення.

Повністю алгоритм відображено на рисунку 3.2.

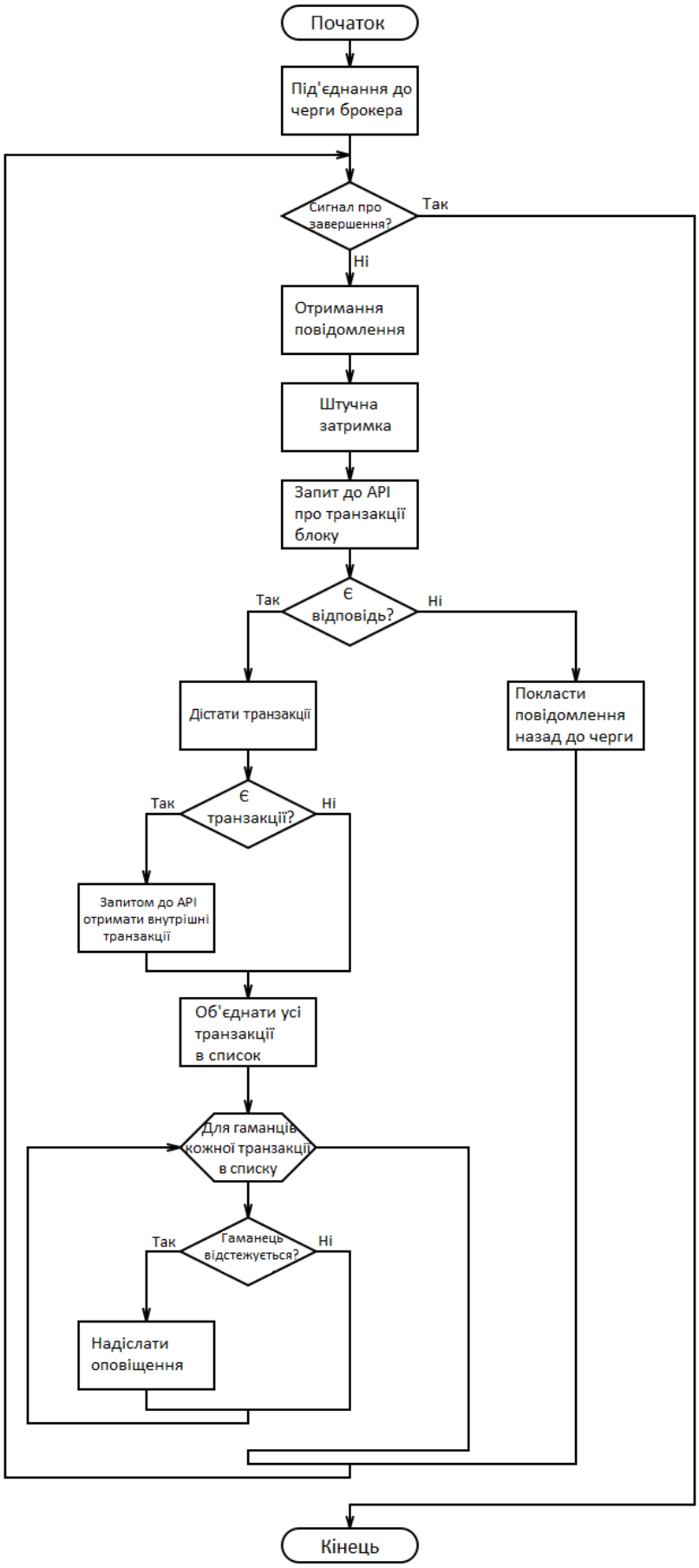


Рис. 3.2. Алгоритм обробки блока

3.2. Розробка інтерфейсу програми

Інтерфейс програми будується на технології чат-ботів месенджеру Telegram. Даний підхід характерний тим, що взаємодія відбувається у вигляді осмисленого діалогу між користувачем та програмним додатком за чітко визначеним алгоритмом – сценарієм. Сценарій фактично є конечним автоматом, оскільки він має свої стани та правила, за якими відбуваються переходи. Також використовується внутрішня пам’ять для збереження проміжних даних.

Інтерфейс розроблюваної програми має надавати користувачу для взаємодії зрозумілий сценарій реєстрації адреси цільової мережі криптовалюти. Під час діалогу додаток має отримати власне адресу для відстежування, а також символьне ім’я (або сформувати його якщо користувач вирішив пропустити цей крок) для відображення інформації, що необхідно в процесі експлуатації.

Даний алгоритм зображено на рисунку 3.3.

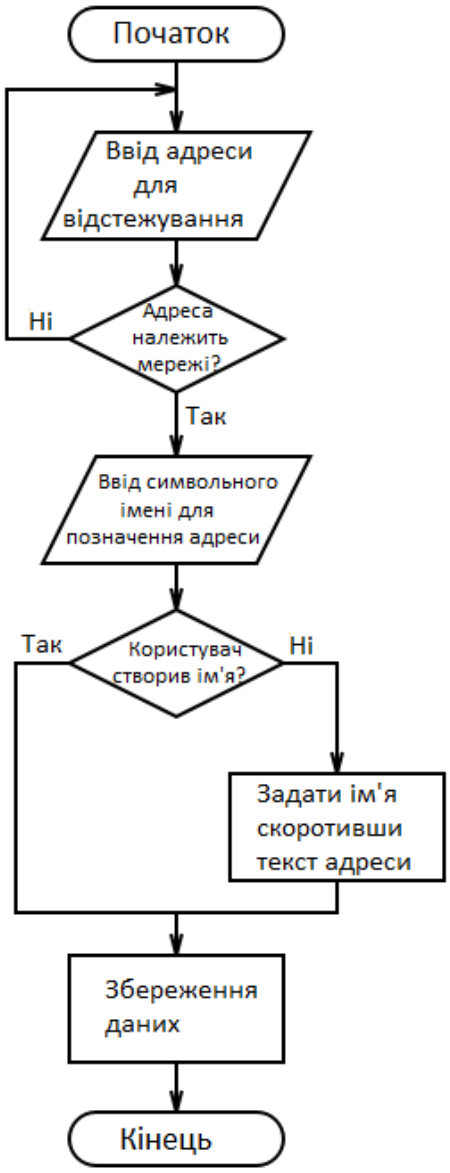


Рис. 3.3. Алгоритм реєстрації адреси

3.3. Проектування структур даних програми

3.3.1. Дані обробки блоків

При обробці блоків виникає необхідність формування повідомлень, що передаються брокером від утиліти взаємодії з мережею до утиліти оповіщення користувачів. Дані повідомлення мають бути достатньо невеликі за об’ємом, оскільки їхнє утворення та накопичення в черзі (у випадку, коли утиліта оповіщення не встигає обробляти блоки) повинно мати мінімальний вплив на

продуктивність системи. Структура таких повідомлень має лише два найбільш необхідні поля, що описано в таблиці 3.1.

Таблиця 3.1 Структура повідомлення

Назва поля	Тип	Призначення
number	int (integer)	Номер отриманого блоку
hash	str (string)	Хеш отриманого блоку

Номер отриманого блоку є ключем для отримання внутрішніх транзакцій блоку, а хеш – для звичайних.

3.3.2. Дані інтерфейсу користувача

Інтерфейс програми для повноцінної взаємодії з користувачем потребує швидкодіюче сховище з принципом зберігання інформації ключ-значення. Оскільки в якості сховища використовується Redis, до якого потрапляють об’єкти json у строковому представленні, доцільно вказати типи даних до конвертації.

Таблиця 3.2 Структура даних інтерфейсу

Назва поля	Тип	Призначення
lang	str (string)	Мова інтерфейсу
wallet	str (string)	Текст адреси
name	str (string)	Символьне ім’я адреси
current_page	list	Історія транзакцій адреси

Інтерфейс додатку на початку має дві мови на вибір: російська та англійська. Обраний варіант зберігається в базі даних та використовується при кожній взаємодії користувача з програмою. Це єдине значення, що має

зберігатися великий проміжок часу, а тому його дублікат також записується до основного сховища.

Поля тексту та імені адреси необхідні лише для зберігання проміжних даних. Тобто, щойно завершується реєстрація адреси, дані записуються до основного сховища, а з бази інтерфейсу видаляються.

Поле історії транзакцій необхідно для мінімізації звернень програми до API сторонніх сервісів. Це не лише допомагає уникнути блокування, але й пришвидшує роботу інтерфейсу. Щойно користувач покидає меню перегляду історії транзакцій – дані видаляються.

3.3.3. Основні дані програми

В основному сховищі зберігаються дані, що підлягають тривалому утриманню. Кількість такої інформації може бути досить великою, а тому база даних має бути легко розширюваною та гнучкою до внесення змін чи корекцій безпосередньо під час роботи програми.

Загалом основне сховище зберігає чотири таблиці з даними:

1. user;

Таблиця user зберігає усіх користувачів програмного додатку та деякі відомості про них.

Таблиця 3.3 user

Назва поля	**Тип**	**Призначення**
user_id	bigint(20)	Унікальний ідентифікатор користувача
reg_date	timestamp	Дата та час реєстрації користувача
lang	varchar(20)	Мова інтерфейсу користувача

Унікальний ідентифікатор присвоюється користувачу месенджером при його ініціюючій реєстрації. Він дозволяє визначити користувача з-поміж інших. У такий спосіб адресуються повідомлення в месенджері. Для визначеної таблиці бази даних він є первинним ключем.

Мова повідомлень додатку зберігається у якості дублікату зі сховища користувацького інтерфейсу. Це дозволяє відновити дані після їх можливої втрати (наприклад, очищення оперативної пам’яті через перезавантаження серверу).

2. wallet;

Таблиця wallet зберігає адреси, зареєстровані користувачами з відповідними супутніми даними.

Таблиця 3.4 wallet

Назва поля	**Тип**	**Призначення**
wallet_id	bigint(20)	Унікальний ідентифікатор адреси
user_id	bigint(20)	Унікальний ідентифікатор користувача
wallet	varchar(100)	Текст адреси
name	varchar(30)	Символьне ім’я адреси
reg_date	timestamp	Дата реєстрації адреси

Унікальний ідентифікатор адреси встановлюється автоматично при додаванні запису до таблиці і є її первинним ключем. Користувач не має доступу до цього ідентифікатору, оскільки це є внутрішня інформація програмного продукту.

Унікальний ідентифікатор користувача відноситься до первинного ключа таблиці user. Разом із текстом адреси цей ідентифікатор утворює

унікальний ключ – користувач не може відстежувати одну і ту ж саму адресу двічі. При спробі додати наявну адресу, відбудеться оновлення її імені та дати реєстрації, однак нового запису створено не буде.

3. history_filter;

Таблиця history_filter призначена для зберігання користувацьких налаштувань стосовно відображення записів історії транзакцій певної адреси мережі криптовалюти.

Таблиця 3.5 history_filter

Назва поля	**Тип**	**Призначення**
filter_id	bigint(20)	Унікальний ідентифікатор фільтру
wallet_id	bigint(20)	Унікальний ідентифікатор адреси
records_number	tinyint(3)	Кількість відображуваних записів в історії транзакцій
show_participant	tinyint(1)	Прапорець відображення іншого учасника транзакції

Унікальний ідентифікатор фільтру є первинним ключем таблиці та встановлюється автоматично. Запис про даний фільтр створюється разом із записом про адресу як результат роботи сценарію реєстрації.

Унікальний ідентифікатор адреси відноситься до первинного ключа таблиці wallet і також є унікальним ключем для даної таблиці. Тобто, неможливо створити два фільтра для відображення записів історії транзакцій однієї адреси.

Кількість відображуваних записів та прапорець відображення учасника – власне фільтри. Кількість записів може коливатися від 0 до 128 рядків, однак

програмно встановлено допустимий діапазон від 1 до 10 включно, а за замовчуванням встановлено значення 5.

Прапорець фактично є булевим значенням. В даному випадку він вказує, чи потрібно відображати від кого або кому надіслано кошти. За замовчуванням встановлено опцію в режим “відображати”.

4. notifications_filter.

У таблиці notifications_filter зберігаються записи щодо налаштування відправки оповіщень користувачу про нові транзакції за участі відстежуваної адреси.

Таблиця 3.6 notifications_filter

Назва поля	**Тип**	**Призначення**
filter_id	bigint(20)	Унікальний ідентифікатор фільтру
wallet_id	bigint(20)	Унікальний ідентифікатор адреси
min_amount	decimal(20, 10)	Мінімальна сума для оповіщення
tr_type	enum('in','out','inout')	Напрямки транзакцій для оповіщення
off_sound	tinyint(1)	Прапорець звукового оповіщення
notify	tinyint(1)	Прапорець увімкнення функції оповіщення

Унікальні ідентифікатори фільтру та адреси виконують аналогічні функції до однойменних полів таблиці history_filter. Також, за аналогією, при створенні запису в таблиці wallet одразу створюється запис для даної адреси в таблиці notifications_filter.

Користувач може налаштувати мінімальну суму за якої йому будуть надходити оповіщення, вказавши її числом у діапазоні від 0.0000000001 до

9999999999.9999999999 не включно. За замовчуванням мінімальна сума відсутня (значення встановлено в 0).

Фільтрувати сповіщення також можна за напрямком транзакції: усі перекази, тільки надходження або лише списання. Для цього використовується тип даних, що має назву перерахування У такий спосіб база даних зберігає одне з трьох вказаних значень. За замовчуванням користувач буде отримувати сповіщення про всі транзакції за участі відстежуваної ним адреси.

Прапорець звукового оповіщення відповідно зберігає один з двох можливих станів, які можна охарактеризувати як “увімкнути” та “вимкнути” звук оповіщень про нові транзакції.

Прапорець увімкнення функції оповіщення дозволяє відмовитися від оповіщень про нові транзакції, не видаляючи при цьому адресу.

3.4. Проектування структури модулів програми

Загальна структура проекту відображена на рисунку 3.4.

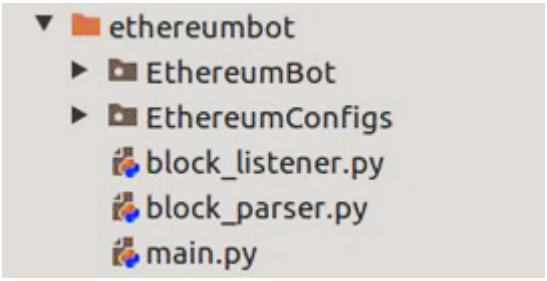


Рис. 3.4. Загальна структура проекту

Розробка має три виконуваних файли:

1. main.py – відповідає за користувацький інтерфейс;
2. block_listener.py – утиліта прийому блоків;
3. block_parser.py – утиліта обробки блоків.

Файл main.py запускає роботу користувацького інтерфейсу, опис якого міститься в директорії EthereumBot. Структура даного каталогу відображена на рисунку 3.5.

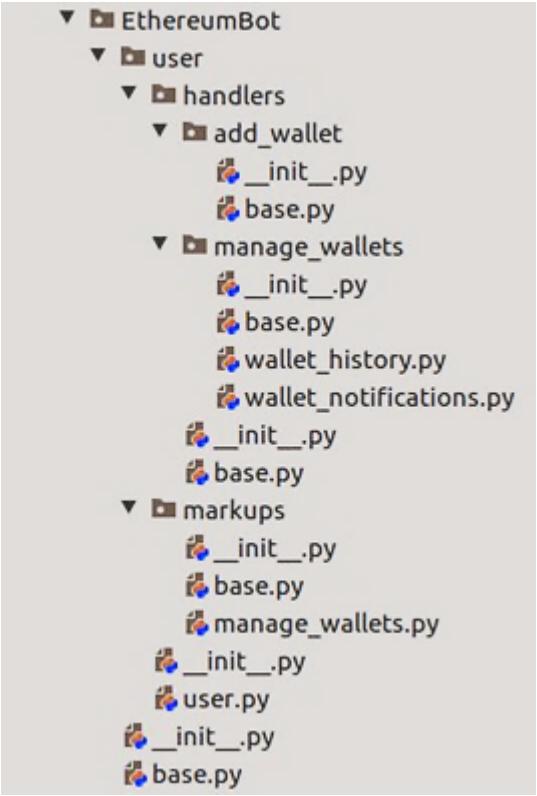


Рис. 3.5. Структура користувацького інтерфейсу

Користувацький інтерфейс складається з трьох основних частин:

1. handlers;
2. markups;
3. user.

Директорія handlers та її піддиректорії містять файли, що обробляють користувацькі запити. Вони формують відповіді на повідомлення чи натискання кнопок. Це головна частина інтерфейсу, до її файлів можуть імпортуватися інші перелічені вище частини.

Директорія markups містить допоміжні функції для формування користувацьких меню – наборів кнопок, за допомогою яких відбувається взаємодія з програмним додатком.

Файл user.py містить клас з переліком станів, яких може набувати роль користувача під час взаємодії з інтерфейсом. Це необхідно для роботи конечних автоматів – сценаріїв.

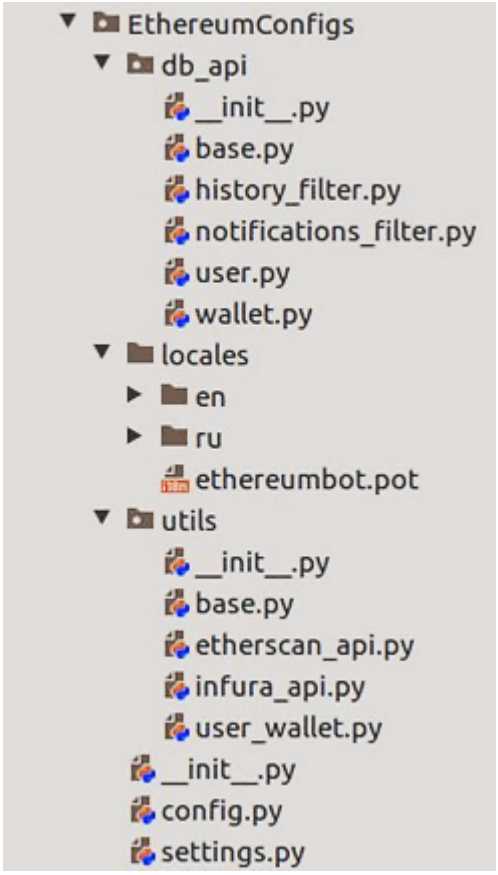


Рис. 3.6. Структура функціонально-конфігураційних файлів

Директорія EthereumConfigs містить модулі, що необхідні для взаємодії інтерфейсів користувача, мережі та бази даних:

1. db_api – містить модулі для взаємодії з базою даних, кожен файл працює з відповідною до назви таблицею;
2. locales – містить переклади користувацького інтерфейсу різними мовами;
3. utils – містить модулі для взаємодії з мережею через API та допоміжні функції для користувацького інтерфейсу.

Також в даному каталозі розміщені конфігураційні файли програмного додатку:

1. config.py – модуль, що містить ключі та приватну інформацію, що необхідна для користування базою даних та API сервісів;
2. settings.py – модуль з базовими налаштуваннями для роботи розроблюваної системи.

Модулі каталогу EthereumConfigs використовуються усіма частинами програмного додатку. Вони імпортуються як до файлів інтерфейсу користувача, так і до підпрограм обробки блоків.

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі було описано процеси розробки та проектування алгоритмів роботи усіх частин розроблюваної системи. Детально описано структури даних, що використовуються програмою.

Результатом роботи є описана в розділі структура виконавчих модулів системи. Спроектована структура є достатньо гнучкою до редагування та розширення функціоналу програмного додатку.

РОЗДІЛ 4

Тестування розробленої системи

4.1. Тестування функціоналу обробки блоків

4.1.1. Опис системи на якій проводиться тестування

Тестування проводиться на віддаленому сервері з наступними технічними характеристиками:

```
*-cpu
     description: CPU
     product: Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40GHz
     vendor: Intel Corp.
     physical id: 401
     bus info: cpu@0
     slot: CPU 1
     size: 2GHz
     capacity: 2GHz
     width: 64 bits
```

Рис. 4.1. Відомості про центральний процесор

```
*-memory
     description: System Memory
     physical id: 1000
     size: 1GiB
     capacity: 1GiB
   *-bank
        description: DIMM RAM
        physical id: 0
        slot: DIMM 0
        size: 1GiB
        width: 64 bits
```

Рис. 4.2. Відомості про об’єм пам’яті

```
Retrieving speedtest.net configuration...
Testing from Digital Ocean
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by Vancis (Amsterdam) [3.12 km]: 1.783 ms
Testing download speed................................
Download: 800.97 Mbit/s
Testing upload speed..................................
Upload: 845.52 Mbit/s
```

Рис. 4.3. Відомості про швидкість з’єднання з мережею Інтернет

4.1.2. Тестування утиліти прийому блоків

Головним показником ефективності роботи підпрограм обробки блоків мережі є їхній час виконання.

Для підпрограми прийому блоків від мережі криптовалюти заміряється час у проміжку від моменту отримання інформації про новий блок системи до моменту завершення операції відправки сформованого повідомлення в чергу брокера.

```
root@botpro:~/Bots/ethereumbot# python3.7 block_listener.py
{"jsonrpc":"2.0","id":1,"result":"0x1"}
b'{"number": 10167018, "hash": "0x087f3474fa8f1247200d5c0df93ec87ec667df3bf2e2ec48d5ea728ed34d7189"}'
10167018: 0.0015141963958740234
b'{"number": 10167019, "hash": "0x8a0ff74c50ef20db6f99c65f19ce6aa428ea75facb0cf0ab7b6f8e5a9faddb3d"}'
10167019: 0.002039670944213867
b'{"number": 10167020, "hash": "0x9237515cc5afe41d9f18ddd8cfe506a3a28e9847182d17139e6f0bfc85c2079e"}'
10167020: 0.0021119117736816406
b'{"number": 10167021, "hash": "0x5341253e8a4c1362bde58023361d63c21ed29fd0ce3b3b314461c31184e7e340"}'
10167021: 0.00205230712890625
b'{"number": 10167022, "hash": "0xe56ee649476a46e9e6eebb98dd2f51344a7270664f6fd52838452dcc155c4d3a"}'
10167022: 0.002168416976928711
b'{"number": 10167023, "hash": "0x5bb20c7f1c295becd782868f5123026127762b85a00b79b4319d8940ac910593"}'
10167023: 0.003706216812133789
b'{"number": 10167024, "hash": "0xc503a17380316369e9b4a64dffd7b79e6c23fe9dc0a44fadf46e063f990196c3"}'
10167024: 0.004320859909057617
b'{"number": 10167025, "hash": "0xb0d1e06cce19f53049f6d2a37bf52fb2f93415d0400f62ffe966e8a5b471bb27"}'
10167025: 0.0018815994262695312
b'{"number": 10167026, "hash": "0xc571dfe3999540c464e79ef8bd7e9ab2a18c9056c25b912f644650f604cbf45e"}'
10167026: 0.0026204586029052734
b'{"number": 10167027, "hash": "0xdc5e2097781daaed21422d723a8f00191b1549c87fa73c7321554ca2583dd2aa"}'
10167027: 0.002067089080810547
b'{"number": 10167028, "hash": "0xbb60345f71d25554e38a9819b48f36f604b12fcf914d16311fe8faf4fe6df572"}'
10167028: 0.0015878677368164062
b'{"number": 10167029, "hash": "0x249a112c5b81acc1045bfcc9fbc289246fbdf3f6677788f3b10f6d07efce72c5"}'
10167029: 0.0020563602447509766
```

Рис. 4.4. Робота утиліти прийому блоків

На «рис. 4.4» зображено скріншот роботи підпрограми. Перший рядок – власне запуск утиліти, другий рядок – підтвердження рукостискання за протоколом WebSocket. Далі серед кожних наступних двох рядків перший з них – це сформоване повідомлення, а другий – час виконання роботи в секундах у форматі “«номер блоку»: «час обробки»”.

Як видно з «рис. 4.4» час роботи підпрограми достатньо малий, аби вдало працювати із високонавантаженою мережею. Порівнюючи отриманий результат із середнім часом реєстрації нових блоків мережі криптовалюти Ethereum (15 секунд), можна сказати, що спроектована утиліта здатна витримати збільшення навантаження у 5000 разів.

4.1.3. Тестування утиліти обробки блоків

Для підпрограми обробки блоків заміряється час від моменту завершення штучної затримки до моменту завершення операції пошуку користувачів та розсилання оповіщень.

```
root@botpro:~/Bots/ethereumbot# python3.7 block_parser.py
10167018: 1.3379912376403809
10167019: 1.5618126392364502
10167020: 0.8002090454101562
10167021: 1.1744771003723145
10167022: 0.5296797752380371
10167023: 0.3106260299682617
10167024: 1.4833745956420898
10167025: 1.6152095794677734
10167026: 0.6115567684173584
10167027: 0.4846358299255371
10167028: 0.4098024368286133
10167029: 0.3872556686401367
```

Рис. 4.5. Робота утиліти обробки блоків

На «рис. 4.5» перший рядок – власне запуск підпрограми, а кожен наступний – час роботи, заміряний в секундах у форматі “«номер блоку»: «час обробки»”.

Як видно з «рис. 4.5» обробка блоків виконується значно повільніше, аніж їхній прийом. Однак, не зважаючи на це, швидкість обробки достатньо висока, а найбільший проміжок часу підпрограма очікує відповіді від API використовуваних сервісів.

Порівнюючи з середнім часом реєстрації блоків у мережі, можна сказати, що утиліта здатна витримати збільшення навантаження у 7 разів, після чого буде мати місце конкурентність підпроцесів обробки блоків. Це стане причиною збільшення навантаження на центральний процесор системи, оскільки, за особливістю асинхронного підходу до написання коду, час очікування відповіді від сервісу для одного блоку буде використаний для внутрішньої обробки іншого.

4.1.4. Результати тестування

Під час роботи програми, сервер має наступний рівень завантаженості:

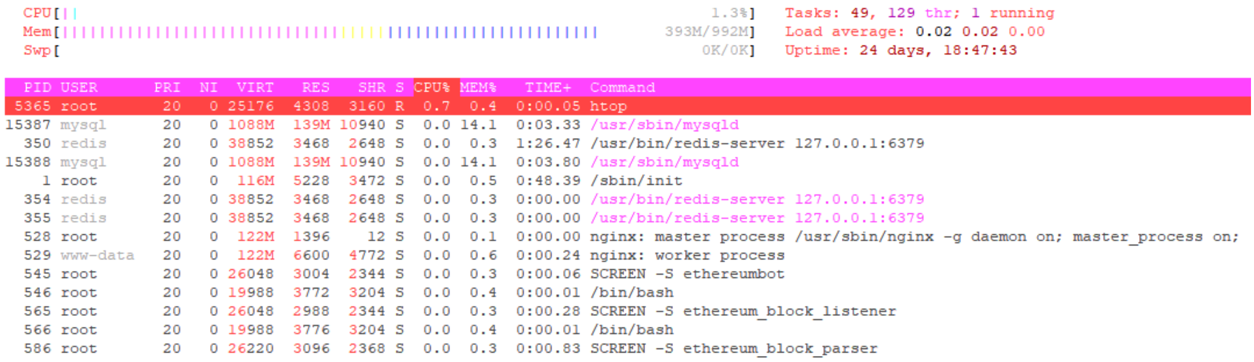


Рис. 4.6. Рівень завантаженості серверу

Оскільки рівень завантаженості не високий, можна зробити висновок, що система написана вдало, а поставлена задача щодо можливості запуску програми на малопотужному комп’ютері виконана.

4.2. Інструкція для користувача інтерфейсу

4.2.1. Початок роботи

Для початку роботи з чат-ботом, необхідно надіслати команду “/start”. Це стандартна процедура для віртуальних співрозмовників месенджеру Telegram. Отримавши дану команду, бот відповідає вітальним повідомленням та надає головне меню:

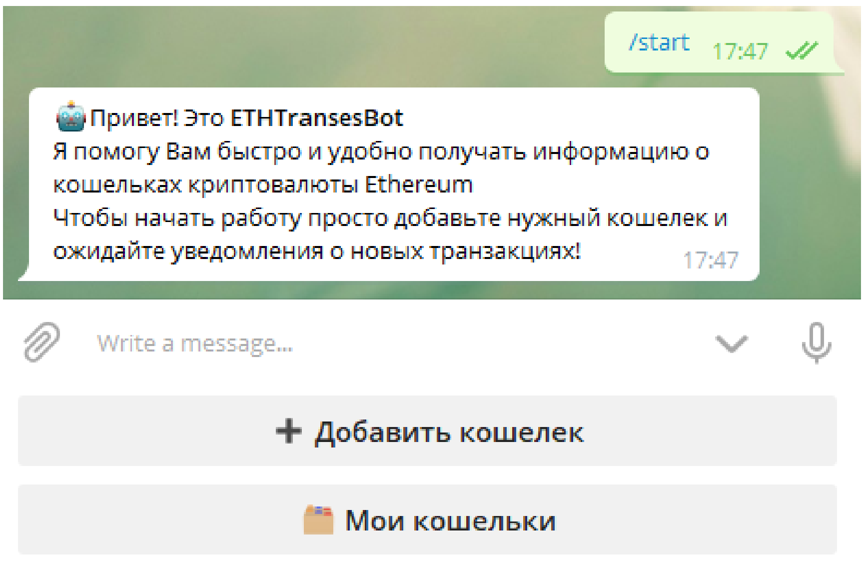


Рис. 4.7. Початок роботи з ботом

4.2.2. Додавання адреси

Щоб додати адресу для відслідковування, необхідно натиснути кнопку “Добавить кошелек”. Чат-бот переходить у стан очікування номеру адреси, а після неї – символьного імені, про що повідомляє користувачу.

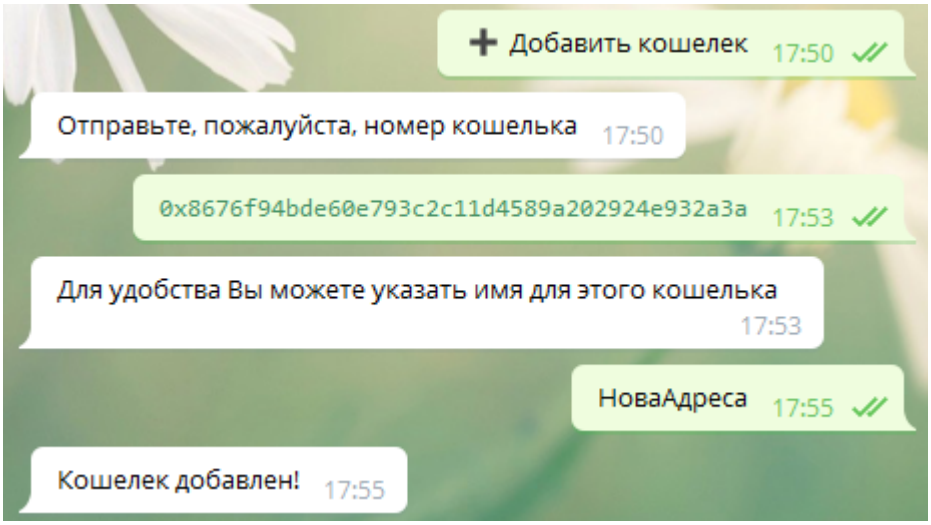


Рис. 4.8. Процес додавання адреси

4.2.3. Перегляд інформації про адресу

Щоб отримати інформацію про відстежувану адресу, необхідно натиснути “Мои кошельки” та вибрати необхідний пункт:

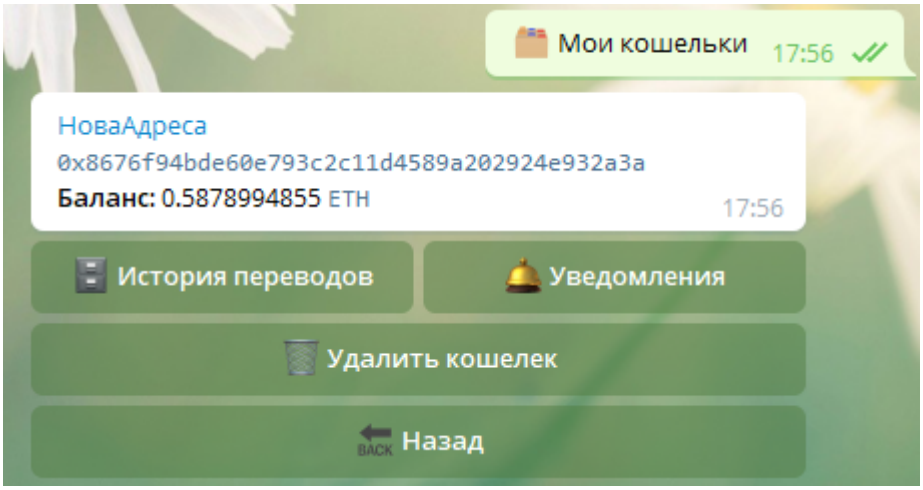


Рис. 4.9. Меню адреси

Відповідно, користувач має можливість переглянути історію транзакцій для різних типів активів:

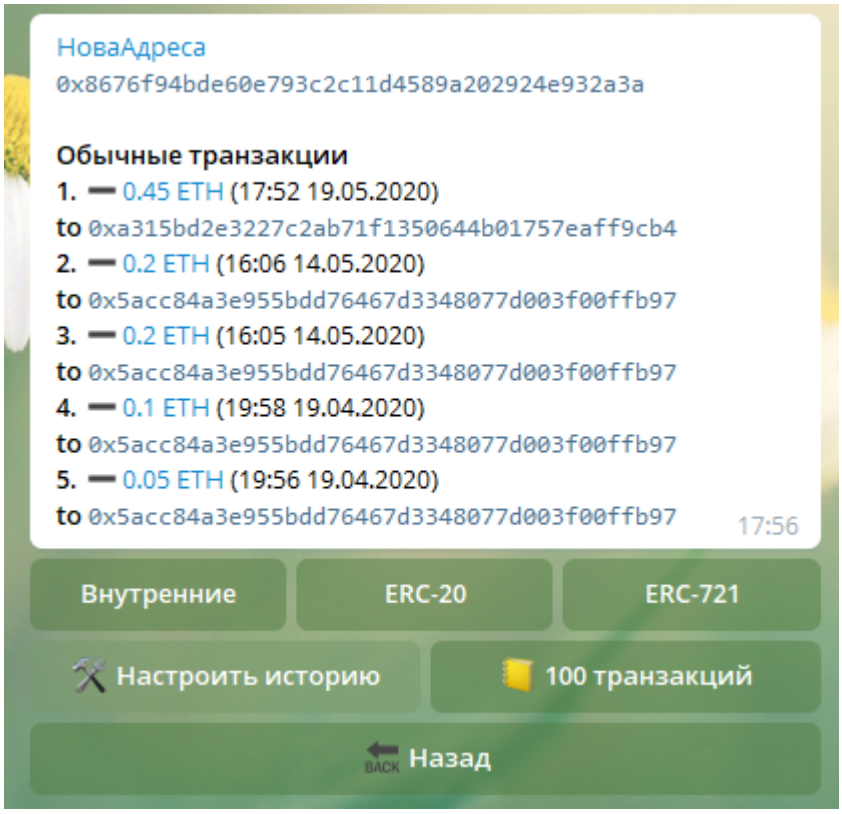


Рис. 4.10. Історія транзакцій адреси

Також можна налаштувати відображення записів історії транзакцій адреси:

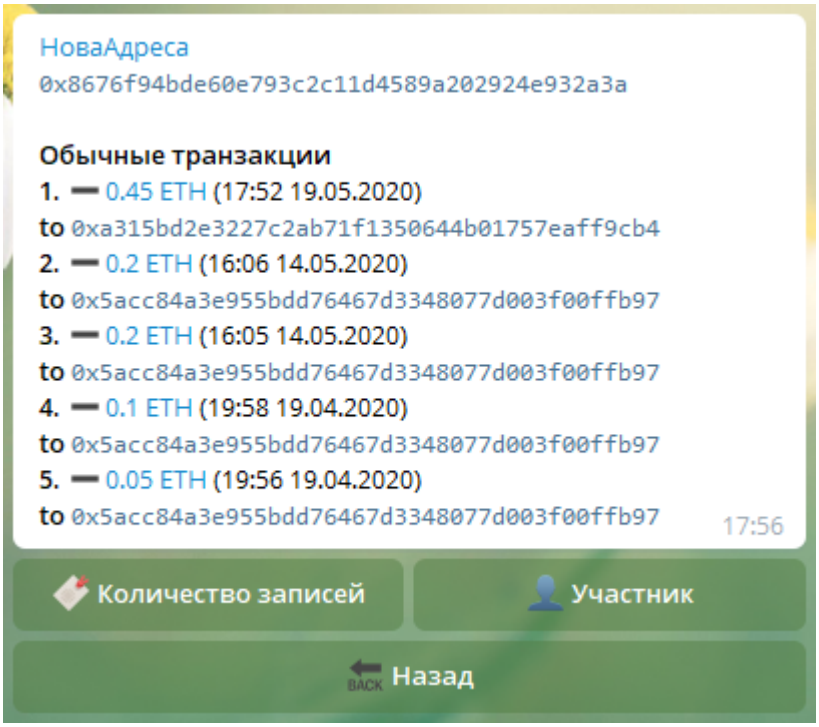


Рис. 4.11. Меню налаштування історії

Також користувач з меню адреси може перейти до меню налаштування повідомлень:

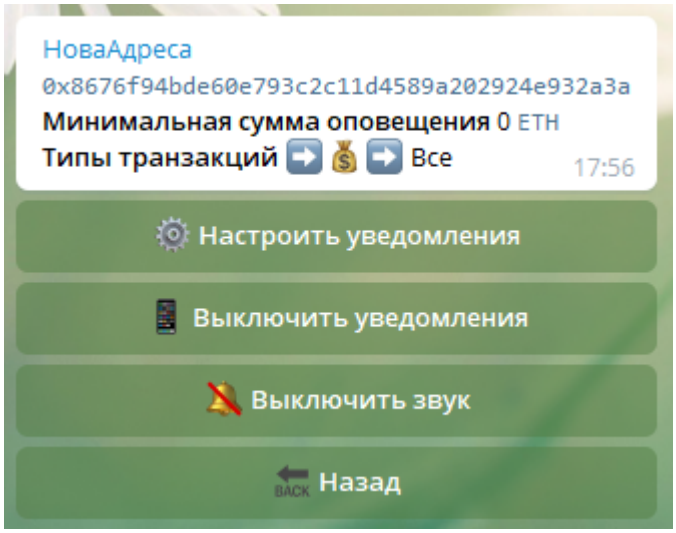


Рис. 4.12. Налаштування повідомлень (перша частина)

У першій частині є кнопки для вимкнення звуку сповіщень або вимкнення сповіщень про нові транзакції адреси взагалі.

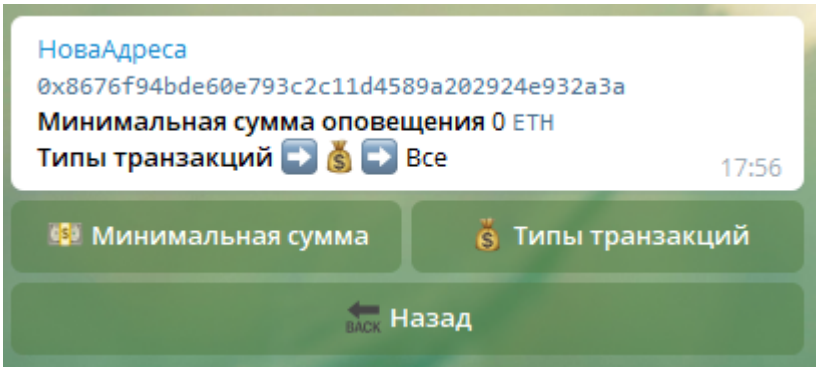


Рис. 4.13. Налаштування повідомлень (друга частина)

У другій частині користувач може встановити мінімальну суму для сповіщення та обрати тип (напрямок) транзакції, про який необхідно сповіщати.

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі було проведено тестування розробленої системи обробки транзакцій мережі криптовалюти Ethereum. Було проведено якісну оцінку програми за допомогою часових замірів виконання її алгоритмів. Результати оцінювання чітко проаналізовано.

Даний розділ надає інструкцію експлуатації для користувача програмного продукту, описує можливості та основні способи взаємодії з наданим інтерфейсом.

ЗАГАЛЬНІ ВИСНОВКИ

Метою дипломного проекту було поставлено створення системи для обробки транзакцій в мережі криптовалюти Ethereum. Також було поставлено ряд задач, які мають бути виконані під час розробки системи.

Для досягнення поставленої мети було опрацьовано та надано у звіті вичерпні теоретичні відомості про цільову предметну область: розглянуто власне поняття криптовалюти, переваги та недоліки криптовалютних мереж, їхні перспективи розвитку. Більш детально розглянуто мережу Ethereum, її особливості, основні складові її вузлів та структуру даних. Окрему увагу приділено технології смарт-контрактів цільової системи, її теперішньому стану та перспективам розвитку.

Проведено короткий огляд та порівняння мереж, що мають схожі властивості до системи Ethereum. Визначено переваги цільової мережі, що обумовили її вибір у якості області розробки.

Було проведено огляд та аналіз ресурсів-аналогів. Визначено їхні особливості, переваги та недоліки. Результати порівняння остаточно сформували способи розробки проекту.

Для досягнення поставленої мети були спроектовані алгоритми роботи усіх частин програми, обрано технології та способи їхньої реалізації. Кожний алгоритм проілюстровано відповідною блок-схемою та наділено поясненням стосовно своїх принципів роботи.

Виконаний проект було протестовано. Тестування проводилося для підпрограм обробки транзакцій із часовими замірами, що є необхідним для оцінки їхньої працездатності. Результати детально проаналізовано та виявлено задовільними.

Зважаючи на аналіз результатів тестування програми, мету проекту досягнуто, а супутні задачі виконано у повному обсязі.

СПИСОК ЛІТЕРАТУРИ

4.1 Таблиця джерел (початок)

№	Джерело
1	Что такое DApps (децентрализованные приложения)? Руководство для начинающих [Онлайновий] // BYTWORK.COM - 06 05 2020 р.. - https://bytwork.com/articles/chto-takoe-dapps-decentralizovannye-prilozheniya-rukovodstvo-dlya-nachinayushchih
2	Криптовалюта [Онлайновий] // Википедия - 13 04 2020 р.. - 02 05 2020 р.. – https://ru.wikipedia.org/wiki/Криптовалюта
3	Криптовалюта [Онлайновий] // Audit-it.ru - 24 07 2017 р.. - 04 05 2020 р.. – https://www.audit-it.ru/terms/accounting/kriptovalyuta.html
4	Курс биткоина преодолел отметку $20000 [Онлайновий] // РБК – 17 12 2017 р.. - 04 05 2020 р.. - https://www.rbc.ru/economics/17/12/2017/5a3667669a79472e25bd5468
5	Что можно купить за биткоин в Украине (рейтинг обновлен в марте 2020) [Онлайновий] // Prosto Coin - 04 05 2020 р.. - https://prostocoin.com/country/ukraine
6	What is the difference between smart contracts and dapps? [Онлайновий] // Quora - 14 03 2017 р.. - 06 05 2020 р.. - https://www.quora.com/What-is-the-difference-between-smart-contracts-and-dapps
7	Что такое смарт-контракты? [Онлайновий] // Хабр - 02 03 2019 р.. - 06 05 2020 р.. - https://habr.com/ru/post/448056/
8	Blockchains and Smart Contracts for the Internet of Things [Онлайновий] // IEEE Xplore® - 10 05 2016 р.. - 06 05 2020 р.. - https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7467408
9	Курс Эфира на сегодня [Онлайновий] // MYFIN.BY - 07 05 2020 р.. - https://myfin.by/crypto-rates/ethereum

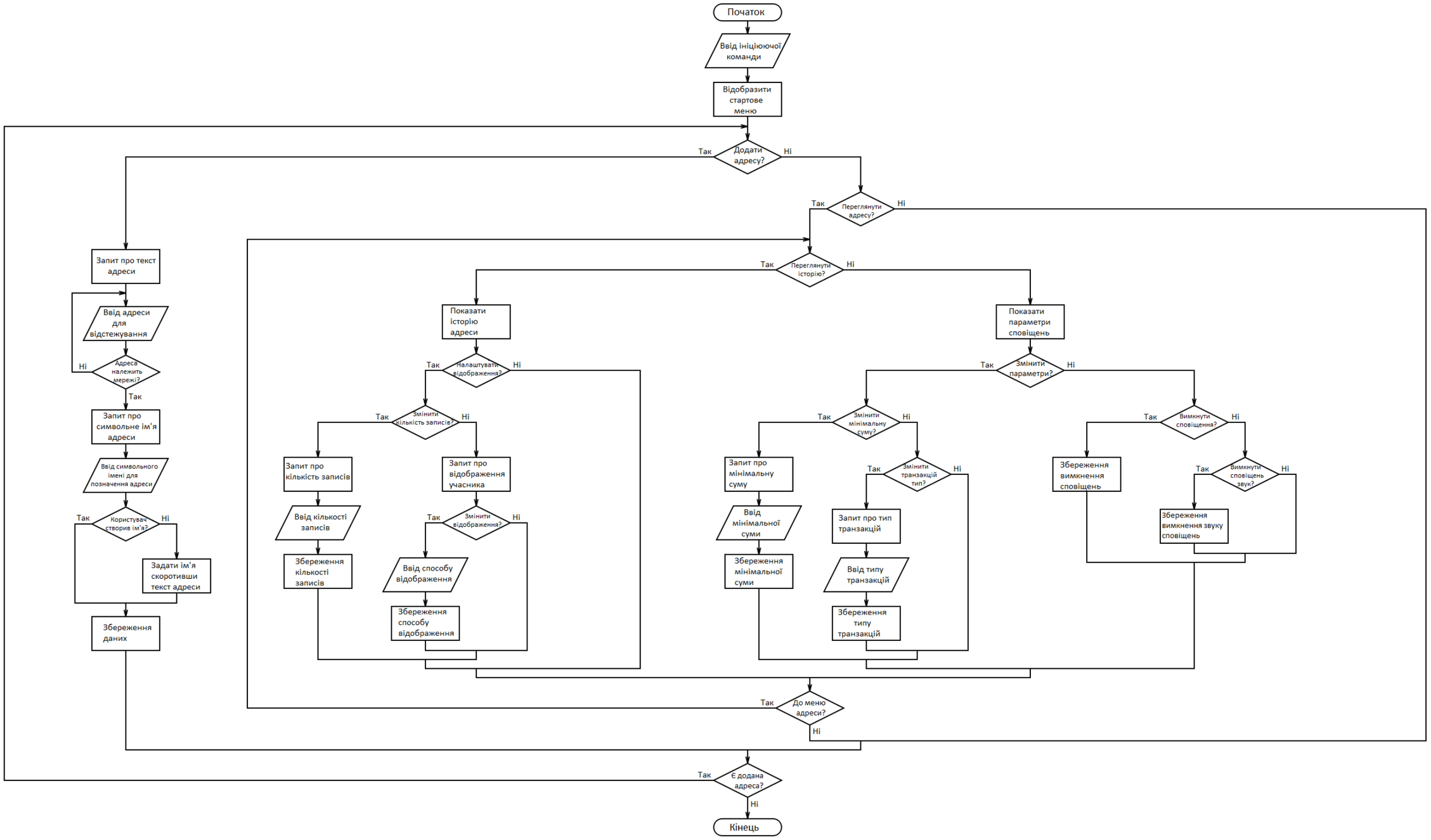
4.1 Таблиця джерел (закінчення)

10	Что такое токены ERC-20? [Онлайновий] // forklog - 03 05 2018 р.. - 06 05 2020 р.. - https://forklog.com/chto-takoe-tokeny-erc-20/
11	What is uPort ? [Онлайновий] // Decrypt - 24 12 2018 р.. - 07 05 2020 р.. - https://decrypt.co/resources/uport
12	A beginner’s guide to Ethereum [Онлайновий] // The Coinbase Blog - 23 02 2017 р.. - 07 05 2020 р.. - https://blog.coinbase.com/a-beginners-guide-to-ethereum-46dd486ceecf
13	What is Ethereum? [Онлайновий] // Ethereum - 11 04 2020 р.. - 07 05 2020 р.. - https://ethereum.org/what-is-ethereum/
14	Введение в смарт-контракты [Онлайновий] // Хабр - 04 06 2018 р.. - 08 05 2020 р.. - https://habr.com/ru/company/distributedlab/blog/413231/
15	Транзакции в сети Ethereum — GAS (газ) комиссии за переводы токенов [Онлайновий] // Майнинг Криптовалюты - 10 05 2020 р.. - https://mining-cryptocurrency.ru/ethereum-tranzakcii-komissii-gas/
16	Алгоритмы Консенсуса в Блокчейне: POW, POS и другие (2020) [Онлайновий] // BYTWORK.COM - 10 05 2020 р.. - https://bytwork.com/articles/algoritmy-konsensusa-v-blokcheyne-powpos-i-drugie#sect5
17	Смарт-контракты и платформы для их реализации [Онлайновий] // DeCenter - 27 02 2018 р.. - 08 05 2020 р.. - https://decenter.org/ru/smart-kontrakty-i-platformy-dlya-ikh-realizatsii
18	Гид по владению аккаунтами и контрактами в Ethereum [Онлайновий] // Хабр - 05 08 2018 р.. - 09 05 2020 р.. - https://habr.com/ru/company/mixbytes/blog/416339/
19	Мессенджеры в Украине: основные игроки, проблемы и перспективы [Онлайновий] // COMMENTS.UA - 09 12 2019 р.. - 11 05 2020 р.. - https://comments.ua/article/it/technology/641679-messendzhery-v-ukraine-osnovnye-igroki-problemy-i-perspektivy.html

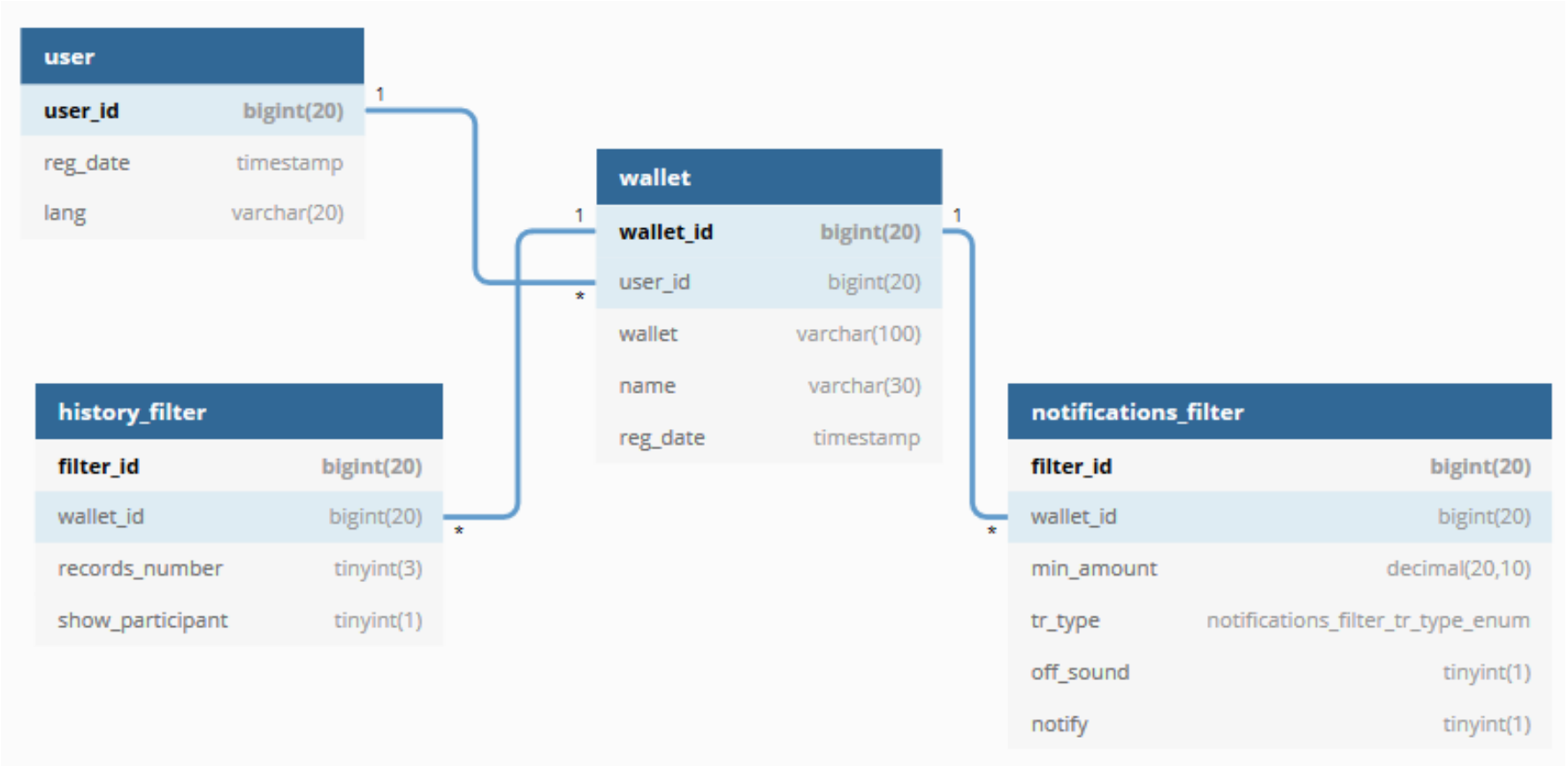
Додаток А
до дипломного проекту

на тему: «Система обробки транзакцій в мережі криптовалюти Ethereum»

Київ – 2020



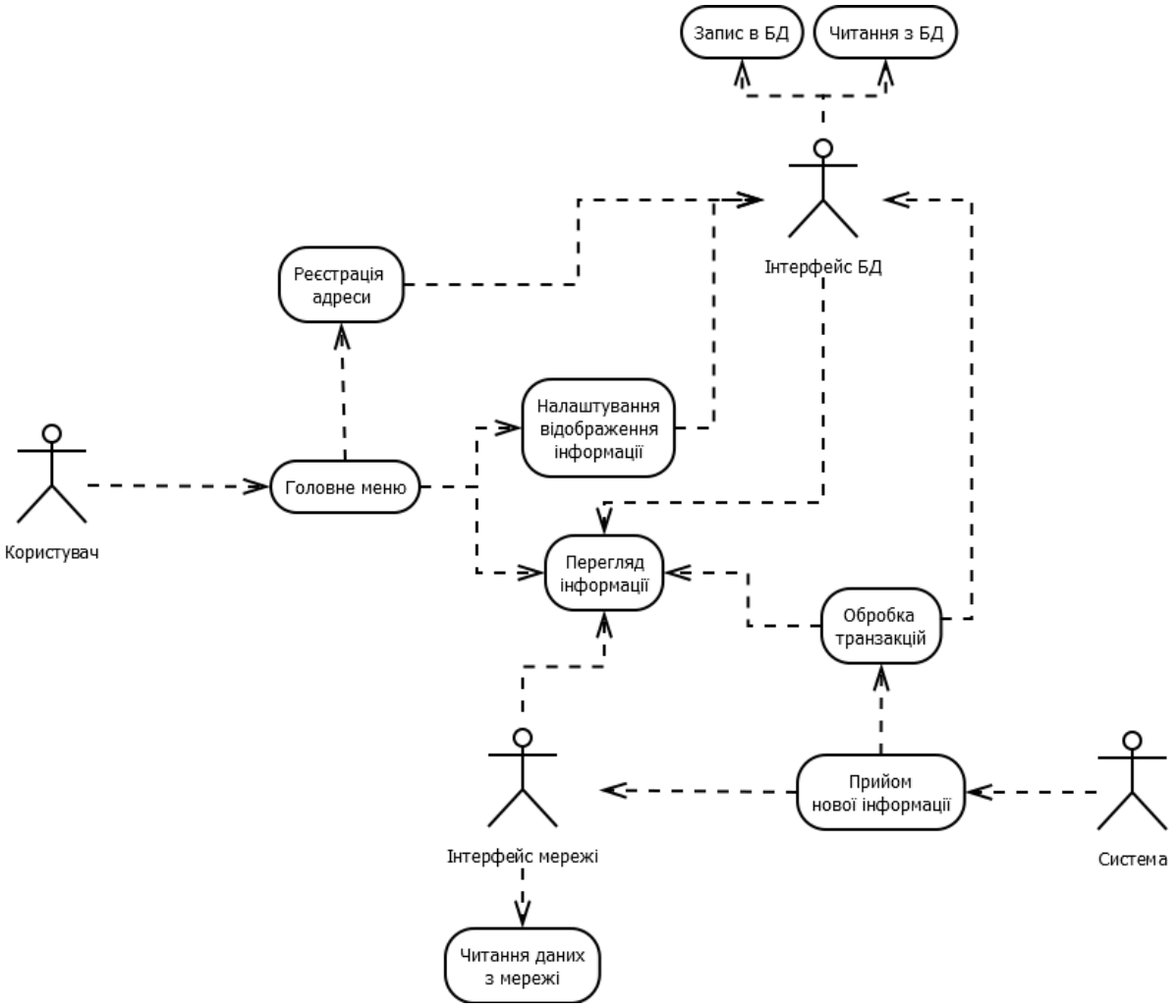
					ДП.7800.04.000 А1
Зм.	Арк.	№ докум.	Підпис	Дата	
Розробив		Дарагань Д.А.			Принципова схема алгоритму
Перевірив		Новотарський М.А.			Літ. / Аркуш 1 / Аркушів 1
Реценз.					НТУУ «КПІ», ФІОТ, ІО-62
Н. Контр.		Сімоненко В.П.			
Затвердив					



					ДП.7800.05.000 А2
Зм.	Арк.	№ докум.	Підпис	Дата	
Розробив		Дарагань Д.А.			Функціональна схема
Перевірив		Новотарський М.А.			
Реценз.					
Н. Контр.		Сімоненко В.П.			
Затвердив					

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ», ФІОТ, ІО-62



					ДП.7800.06.000 А3	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		*Дарагань Д.А.*			*Структурна схема*	Літ. / Аркуш 1 / Аркушів 1
Перевірив		*Новотарський М.А.*				
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-62*
Н. Контр.		*Сімоненко В.П.*				
Затвердив						

Додаток Б
до дипломного проекту

на тему: «Система обробки транзакцій в мережі криптовалюти Ethereum»

Київ – 2020

block_listener.py

```
import asyncio
import json

from aio_pika import connect, Message
import websockets

from EthereumConfigs import INFURA_PROJECT_ID


request = {
    'new_heads': {
        'jsonrpc': '2.0',
        'id': 1,
        'method': 'eth_subscribe',
        'params': ['newHeads']
    }  # get hash
}


def get_block_hash(block):
    block_hash = json.dumps({
        'number': int(block['params']['result']['number'], 16),
        'hash': block['params']['result']['hash']
    })
    return block_hash.encode()


async def main():
    # RabbitMQ connection
    async with await connect("amqp://guest:guest@localhost/", loop=loop) as
connection:
        # Create a channel
        channel = await connection.channel()
        # Declaring queue (if it's not declared yet)
        queue = await channel.declare_queue('block_hash_queue')

        # INFURA connection
        async with
websockets.connect(f'wss://mainnet.infura.io/ws/v3/{INFURA_PROJECT_ID}') as
ws_head:
            # Send hello message
            await ws_head.send(json.dumps(request['new_heads']))
            print((await ws_head.recv()))

            # Listen to new blocks
            while True:
                try:
                    # Get new block head
                    new_head = json.loads((await ws_head.recv()))

                    # Get block hash
                    block_hash = get_block_hash(new_head)
                    print(block_hash)
```

```
                    # Send block hash to process
                    await
channel.default_exchange.publish(Message(block_hash), routing_key=queue.name)
                except websockets.exceptions.ConnectionClosed:
                    print('ConnectionClosed')
                    break


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
    loop.close()
```

block_parser.py

```
import asyncio
import json

from aio_pika import connect, Message, IncomingMessage
from aiogram import exceptions
from web3 import Web3

from EthereumConfigs import bot, i18t, get_lang, db_api, utils


request = {
    'get_block': {
        'jsonrpc': '2.0',
        'method': 'eth_getBlockByHash',
        'params': ['', True],
        'id': 2
    }
}

LAST_BLOCK = 0


async def get_users_to_notify(cur: db_api.Cursor, wallet: str, amount: float,
tr_type: str) -> tuple:
    await cur.execute('select w.user_id, w.name, nf.off_sound '
                      'from wallet w join notifications_filter nf on
w.wallet_id = nf.wallet_id '
                      'where w.wallet = %s and nf.notify = 1 and nf.min_amount
>= %s and nf.tr_type <> %s',
                      (wallet, amount, tr_type))
    users = await cur.fetchall()
    return users


async def process_parse_transactions(transactions):
    con, cur = await db_api.create_con()

    for tr in transactions:
```

```
        value = float(Web3.fromWei(int(tr['value'], 16), 'ether'))

        for user in await get_users_to_notify(cur, tr['from'], value, 'in'):
            user_id, name, off_sound = user
            try:
                await bot.send_message(user_id,
                                       i18t('withdraw_message', locale=(await
get_lang(user_id))).format(
                                           tr['from'], name, tr['hash'],
value, tr['to']
                                       ),
                                       disable_web_page_preview=True,
                                       disable_notification=off_sound)
            except exceptions.BotBlocked:
                pass

        for user in await get_users_to_notify(cur, tr['to'], value, 'out'):
            user_id, name, off_sound = user
            try:
                await bot.send_message(user_id,
                                       i18t('replenishment_message',
locale=(await get_lang(user_id))).format(
                                           tr['to'], name, tr['hash'], value,
tr['from']
                                       ),
                                       disable_web_page_preview=True,
                                       disable_notification=off_sound)
            except exceptions.BotBlocked:
                pass

    await con.ensure_closed()


async def get_internal_transactions(block_number, tr_number):
    transactions = await utils.get_block_internal_transactions(block_number,
tr_number)
    for in_tr in transactions:
        in_tr['value'] = hex(int(in_tr['value']))
    return transactions


def grab_transactions(channel, queue):
    async def on_message(message: IncomingMessage):
        global LAST_BLOCK, request

        # Get block hash from queue
        block_hash = json.loads(message.body.decode())

        # Buffer time needed to wait until the block will be registered on the
network
        await asyncio.sleep(60)

        # Avoid parsing same blocks if error occurs
        if block_hash['number'] <= LAST_BLOCK:
```

```
            return

        # Perform request
        response = await utils.get_block_transactions(block_hash['hash'])

        # Get result from received block
        result = response.get('result')

        # Put block back to the queue if it's not registered yet
        if not result:
            await channel.default_exchange.publish(
                Message(json.dumps(block_hash).encode()),
                routing_key=queue.name
            )
            return

        # Parse received transactions
        transactions = result.get('transactions', [])

        # Add internal transactions
        if transactions:
            internal_transactions = await
get_internal_transactions(block_hash['number'], len(transactions))
            transactions.extend(internal_transactions)

        # print(transactions)

        # Process transactions
        await process_parse_transactions(transactions)

        LAST_BLOCK = block_hash["number"]
    return on_message


async def main():
    # RabbitMQ connection
    connection = await connect("amqp://guest:guest@localhost/", loop=loop)
    # Creating a channel
    channel = await connection.channel()
    # Declaring queue
    queue = await channel.declare_queue('block_hash_queue')
    # Start listening the queue
    await queue.consume(grab_transactions(channel, queue), no_ack=True)


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.create_task(main())
    loop.run_forever()
```

etherscan_api.py

```
from urllib.parse import urlencode
```

```
from web3 import Web3

from EthereumConfigs import ETHERSCAN_TOKEN
from .base import fetch_content_get, get_transactions


ETHERSCAN_ENDPOINT = 'https://api.etherscan.io/api'


async def get_user_balance(wallet: str) -> str:
    params = {
        'module': 'account',
        'action': 'balance',
        'address': wallet,
        'tag': 'latest',
        'apikey': ETHERSCAN_TOKEN
    }
    url = f'{ETHERSCAN_ENDPOINT}?{urlencode(params)}'
    response = await fetch_content_get(url)
    balance = f'{float(Web3.fromWei(int(response["result"]),
"ether")):.10f}'.rstrip('0').rstrip('.')
    return balance


async def get_wallet_transactions(wallet: str, records_number: int = 5) ->
list:
    params = {
        'module': 'account',
        'action': 'txlist',
        'address': wallet,
        'page': 1,
        'offset': records_number,
        'sort': 'desc',
        'apikey': ETHERSCAN_TOKEN
    }
    url = f'{ETHERSCAN_ENDPOINT}?{urlencode(params)}'
    transactions = await get_transactions(url)
    return transactions


async def get_wallet_internal_transactions(wallet: str, records_number: int =
5) -> list:
    params = {
        'module': 'account',
        'action': 'txlistinternal',
        'address': wallet,
        'page': 1,
        'offset': records_number,
        'sort': 'desc',
        'apikey': ETHERSCAN_TOKEN
    }
    url = f'{ETHERSCAN_ENDPOINT}?{urlencode(params)}'
    transactions = await get_transactions(url)
    return transactions
```

```
async def get_wallet_erc20_transactions(wallet: str, records_number: int = 5)
-> list:
    params = {
        'module': 'account',
        'action': 'tokentx',
        'address': wallet,
        'page': 1,
        'offset': records_number,
        'sort': 'desc',
        'apikey': ETHERSCAN_TOKEN
    }
    url = f'{ETHERSCAN_ENDPOINT}?{urlencode(params)}'
    transactions = await get_transactions(url)
    return transactions


async def get_wallet_erc721_transactions(wallet: str, records_number: int = 5)
-> list:
    params = {
        'module': 'account',
        'action': 'tokennfttx',
        'address': wallet,
        'page': 1,
        'offset': records_number,
        'sort': 'desc',
        'apikey': ETHERSCAN_TOKEN
    }
    url = f'{ETHERSCAN_ENDPOINT}?{urlencode(params)}'
    transactions = await get_transactions(url)
    return transactions


async def get_block_internal_transactions(block_number: int, tr_number: int) -
> list:
    params = {
        'module': 'account',
        'action': 'txlistinternal',
        'startblock': block_number,
        'endblock': block_number,
        'page': 1,
        'offset': tr_number,
        'sort': 'asc',
        'apikey': ETHERSCAN_TOKEN
    }
    url = f'{ETHERSCAN_ENDPOINT}?{urlencode(params)}'
    response = await fetch_content_get(url)
    return response['result']
```

infura_api.py

```
from json import dumps
```

```
from EthereumConfigs import INFURA_PROJECT_ID
from .base import fetch_content_post


INFURA_ENDPOINT = f'https://mainnet.infura.io/v3/{INFURA_PROJECT_ID}'


async def get_block_transactions(block_hash: str):
    data = {
        'jsonrpc': '2.0',
        'method': 'eth_getBlockByHash',
        'params': [block_hash, True],
        'id': 2
    }
    response = await fetch_content_post(INFURA_ENDPOINT, data=dumps(data))
    return response
```